

ARBEITSBERICHTE DES INSTITUTS FÜR MATHEMATISCHE MASCHINEN UND DATENVERARBEITUNG (INFORMATIK)

FRIEDRICH ALEXANDER UNIVERSITÄT ERLANGEN NÜRNBERG

**HERAUSGEBER : H. Billing, W. Händler, U. Herzog,
F. Hofmann, K. Leeb, P. Mertens, H. Müller, G. Nees,
H. Niemann, D. Seltzer, H. J. Schnelder**

Die optimale Zerlegung von Netzen
und die Optimierung des Prozeßverhaltens
im Adreßraum

Herman Widjaja
—

Dissertation

BAND 12 · NUMMER 8 · ERLANGEN · OKTOBER 1979

1979 #332



Digitized by the Internet Archive
in 2026

https://archive.org/details/IA41546211_0104

ARBEITSBERICHTE DES INSTITUTS FÜR MATHEMATISCHE MASCHINEN UND DATENVERARBEITUNG (INFORMATIK)

FRIEDRICH ALEXANDER UNIVERSITÄT ERLANGEN NÜRNBERG

**HERAUSGEBER : H. Billing, W. Händler, U. Herzog,
F. Hofmann, K. Leeb, P. Mertens, H. Müller, G. Nees,
H. Niemann, D. Seltzer, H. J. Schnelder**

Die optimale Zerlegung von Netzen und die Optimierung des Prozeßverhaltens im Adreßraum

Dem Fachbereich Ingenieurwissenschaften
der Universität Erlangen-Nürnberg
zur Erlangung des Grades
DOKTOR - INGENIEUR

vorgelegt von
Herman Widjaja
(genannt Hoo Chou Ang)

Erlangen, 1979

Dissertation

ARBEITSBERICHT
DES INSTITUTS FÜR MATHEMATIK
MASCHINEN UND DATENVERARBEITUNG
(INFORMATIK)
VERGLEICH ALGEBRAISCHER UND LOGISCHER SYSTEME
VON: DR. W. HÄNDLER
LEITUNG: DR. P. P. SPIES
JAHRE: 1978/79

Die folgende Erklärung vorliegend
und der Genehmigung der Fakultät
in Erlangen



Als Dissertation genehmigt vom
Fachbereich Ingenieurwissenschaften
der Universität Erlangen-Nürnberg

Tag der Einreichung: 22 September 1978
Tag der Promotion: 25 Juli 1979

Dekan: Prof. Dr. D. Seitzer
Berichterstatter: Prof. Dr. W. Händler
Prof. Dr. P. P. Spies

A b s t r a c t

Widjaja, Herman:

DIE OPTIMALE ZERLEGUNG VON NETZEN UND DIE OPTIMIERUNG DES PROZEßVERHALTENS IM ADREßRAUM

In dieser Arbeit wird zunächst das Problem der optimalen Zerlegung von Netzen untersucht. Unter einem Netz wird ein endlicher, zusammenhängender schlichter Graph verstanden, dessen Knoten und dessen Kanten gewichtet sind. Das Problem besteht darin, ein Netz so in Teilgraphen zu zerlegen, daß das Gesamtgewicht der Kanten zwischen den Teilgraphen minimiert wird unter der Bedingung, daß das Gesamtgewicht der Knoten in jedem Teilgraphen eine vorgegebene Gewichtsgrenze nicht übersteigt. Ein mathematisch begründetes, praktisch brauchbares, neues Rechenverfahren wird entwickelt, das exakte Lösungen des Problems liefert. Das Verfahren beruht auf einem Konzept, bei dem das Problem in mehrere, einfacher zu lösende Teilprobleme zerlegt wird. Diese werden nacheinander gelöst; die Lösungen werden als gegebene Parameter des darauffolgenden Teilproblems benutzt. Die Lösung des eigentlichen Problems ergibt sich dann aus der Lösung des letzten Teilproblems. Die wesentlichen Rechenschritte dieses Rechenverfahrens bestehen aus Suchoperationen und logischen Verknüpfungen von boolischen Vektoren. Deshalb eignet es sich besonders gut für die Implementierung in einer Rechenanlage. Ein zusätzlicher Vorteil liegt in der Flexibilität des Verfahrens bei der Lösung sehr großer Probleme, bei denen eine approximierende Lösung mit gegebenen bzw. mit berechneten Fehlerschranken innerhalb der zur Verfügung gestellten Rechenzeit gefunden werden soll.

Für die Untersuchung des Prozeßverhaltens im Adreßraum wird ein Prozeßmodell gebildet, das auf dem speicherorientierten Programmgraphen (kurz, Programmgraphen) beruht. Der Programmgraph ist ein Netz, dessen Knoten sowohl Programmabschnitte als auch Datenstrukturen darstellen. Der Ablauf des Prozesses wird als eine Tour durch den Adreßraum desjenigen Programms aufgefaßt, auf das sich der Prozeß bezieht. Das Prozeßverhalten wird durch eine endliche Folge von Aktivitätsphasen beschrieben, in denen verschiedenartige Aktivitäten vom entsprechenden Prozeß ausgelöst werden. In den Aktivitätsphasen sind Informationen enthalten, die zur Bestimmung der Parameter des Programmgraphen benutzt werden können. Es stellt sich heraus, daß die Optimierung des Prozeßverhaltens im Adreßraum identisch ist mit der optimalen Zerlegung von Programmgraphen.

Abstract

The problem considered in this thesis is that of partitioning an edge- and node-weighted, connected simple graph (called a network) into subgraphs, each of which has a total node-weight that does not exceed a given maximum (i.e. a weight constraint). The objective of partitioning is to assign nodes to subgraphs so that the total weight of edges interconnecting the subgraphs is minimized under this constraint.

This thesis presents first of all a mathematically founded, effective new method for solving this problem, which is fast enough to be practical in finding exact solutions of large problems. The procedure is based on the concept of "divide and conquer", by means of which the problem is partitioned into several subproblems that can be solved efficiently one after the other. The solutions of a subproblem are used as given parameter for the next subproblem, and the procedure proceeds until the last subproblem has been solved. The solutions of the last subproblem provide the result required.

The computation involved in the procedure consists mainly of systematic search and logical operations on binary vectors. Therefore, it is very suitable for operating in a digital computer. In dealing with very large problems under restricted computing time and/or memory space, this new method is flexible enough to produce an approximate solution with error bounds.

In the second part of this thesis the process behavior in address space is investigated. For this purpose a formal process model based on storage oriented program graph is presented, which is essentially a network whose nodes represent linear program segments or any data structures. Each process can be considered as making a "tour" through the storage oriented program graph of a running program, the dynamic behaviour of which is described by a sequence of activity-phases initiated by the corresponding process. The activities in each phase are described by the data access and/or segment transition frequencies, which in turn can be used for calculating the edge- and node-weights of the storage oriented program graph. The second part concludes with a theorem stating that the optimization of process behaviour in address space and the optimal partitioning of storage oriented program graph are identical problems.

<u>Inhaltsverzeichnis</u>	<u>Seite</u>
1. Einleitung	2
1.1 Vorbemerkungen und Motivation	2
1.2 Optimale Zerlegung von Netzen - Einführung in die Problematik	9
1.3 Allgemeine Bemerkungen zur Optimierung des Prozeßverhaltens im Adreßraum	13
2. Die optimale Zerlegung von Netzen - exakte Lösung	15
2.1 Das Problem	15
2.2 Das Angsche Konzept zur Lösung des Problems	30
2.3 Der Blockindikator, die Indikatormenge und die Indikatorfamilien	42
2.4 Der binäre Verzweigungsprozeß	59
2.5 Der Erweiterungsprozeß	90
2.6 Der Ergänzungsprozeß	121
2.7 Eine zusammenfassende Darstellung des Angschen Rechenverfahrens zur optimalen Zerlegung von Netzen	134
2.8 Problembeispiele und Bemerkungen zum Rechenaufwand	136
3. Die optimale Zerlegung von Netzen - deterministische Approximation	139
3.1 Einführung	139
3.2 Ein Annäherungsverfahren, bei dem der relative Fehler vorgegeben ist	139
3.3 Binärer Verzweigungsprozeß mit vorgegebener Rechenzeit	144
3.4 Annäherungsverfahren mit abgekürzter K-Liste	148
3.5 Probleme mit einer oberen und einer unteren Gewichtsgrenze	151

4. Die Optimierung des Prozeßverhaltens im Adreßraum	154
4.1 Speicher, Adreßraum, Speicherhierarchie und virtueller Speicher	154
4.2 Programmschemata und speicherorientierte Programmgraphen	163
4.3 Dynamisches Prozeßverhalten im Adreßraum	181
4.4 Ein Konzept zur Optimierung des Prozeßverhaltens im Adreßraum	191
4.5 Schlußbemerkung	195
5. Literatur	196
Anhang A: Allgemeine Bezeichnungen	217
Anhang B: Graphentheoretische Grundbegriffe	218
Anhang C: Werte von $P(n)$ und 2^n	222
Anhang D: Problembeispiele - exakte Lösung	223
Anhang E: Beispiele für die deterministische Approximation	234

1. Einleitung

1.1 Vorbemerkungen und Motivation

Mit dem rasch zunehmenden Entwicklungstempo der Hardware-Technologie /222/ werden immer mehr neue Rechensysteme heterogener bzw. innovativer Architektur /232,233,243,253/ konzipiert und realisiert, von denen enorme Leistungen zur Lösung sowohl allgemeiner Probleme als auch spezieller komplexer Rechenaufgaben erwartet bzw. abgefordert werden. Solche modernen Rechensysteme besitzen ein gemeinsames charakteristisches Merkmal: Ihre Hardware-Struktur besteht aus einem Netz /265/ von Prozessoren und Speichern, welche gemäß wohldefinierter Vorschriften durch den Austausch von Nachrichten miteinander kommunizieren können /227,228/. Die Prozessoren und Speicher brauchen funktionsmäßig bzw. strukturmäßig nicht miteinander identisch zu sein. Je nach der Aufgabenstellung übernehmen manche Prozessoren z.B. die Funktion eines Allzweckprozessors, manche andere bekommen spezielle Aufgaben zugeteilt; eine Teilmenge dieser Prozessoren und Speicher lassen sich sogar zu einem Großrechner spezieller Struktur, wie z.B. Feldrechner, zusammenfassen /231/. Um die Arbeiten der Prozessoren und Speicher in einem solchen Netz wirksam und fehlerfrei zu organisieren und zu koordinieren, benötigt man ein System von asynchron ablaufenden, miteinander kooperierenden Prozessen /225,226,237,254/, die den Rechenbetrieb im Netz aufrechterhalten. Jeder Prozeß dieses Prozeßsystems hat die Aufgabe, einen bestimmten Typ von Betriebsmitteln zu verwalten oder einen speziellen Dienst des Rechenbetriebes zu leisten. Als Betriebsmittel betrachten wir sowohl die Bestandteile des Netzes, wie z.B. Prozessoren, Verbindungswege zwischen den Prozessoren, Speicherbereiche usw., als auch alle Mittel zum Zweck der Kommunikation wie z.B. Nachrichten, Warteschlangen usw.. Als Dienstleistungen des Rechenbetriebes werden alle Dienste an Benutzer, wie z.B. Laden, Compilieren und Binden von Programmen, Dateiverwaltung usw., bezeichnet. Das Prozeßsystem, in dem jeder Prozeß eine der eben erwähnten Aufgaben zugeteilt bekommt, wird Betriebssystem /235/ genannt. Moderne Betriebssysteme besitzen eine hierarchische Struktur /224,236,249,251/, wobei jeder Prozeß je nach dessen Aufgabe eine Stellung, die als "Schicht" bezeichnet wird, in der Hierarchie bezieht. Alle Prozesse der gleichen Schicht sind in der Lage, durch Austausch von Nachrichten miteinander zu kommunizieren bzw. durch Forderung gemeinsamer Betriebsmittel gegenseitig zu beeinflussen. Die Prozesse einer unteren Schicht werden den Prozessen der oberen Schicht als virtuelle Betriebsmittel zur Verfügung gestellt. Jede Schicht des Betriebssystemes wird aus diesem Grunde als ein Netz von virtuellen Betriebsmitteln aufgefaßt.

Aus der Sicht der Benutzer stellt ein modernes Rechensystem einen wohl-organisierten Betrieb dar, der den Benutzern verschiedenartige Dienste anbietet. Solche Dienste werden in der Regel von den Prozessen der oberen Schicht des Betriebssystems angeboten, während die Prozesse der unteren Schicht anonym im Hintergrund bleiben. Jeder Benutzer ist berechtigt, bestimmte Dienstleistungen vom Betriebssystem abzuverlangen, in dem er ihm eine Forderung stellt. Diese Forderung wird Auftrag genannt. Wird ein Auftrag vom Rechensystem übernommen, dann werden die zur Ausführung des Auftrages notwendigen Betriebsmittel bereitgestellt. Wird der Auftrag ausgeführt, so nimmt er die Dienst anderer Prozesse des Rechensystems in Anspruch. Aus der Sicht des Betriebssystems stellt also jeder Auftrag einen übergeordneten Prozeß dar, welcher Benutzerprozeß genannt wird. Aus der Menge aller Aufträge, welche die Benutzer dem Rechensystem erteilt haben, entsteht nun ein System von asynchron ablaufenden, miteinander kooperierenden Benutzerprozessen, die zusammen mit den dienstleistenden Prozessen des Rechensystems eine "Gesellschaft von Prozessen" bilden. Informell kann man das Verhalten eines Prozesses in dieser Gesellschaft als eine Folge von Aktivitäten im Rechensystem auffassen, wobei die Aktivität des Prozesses zu einem gegebenen Zeitpunkt die Menge der von ihm zu diesem Zeitpunkt angeforderten Betriebsmittel kennzeichnet. Die Leistungsfähigkeit dieser Gesellschaft wird einerseits von ihrer Struktur, ihrer Organisation und ihren Verhaltensnormen, andererseits von der Leistung und vom Verhalten des einzelnen Prozesses bestimmt. Analog zum Verhalten der Menschen in ihrer Gesellschaft kann man sich das Verhalten der Prozesse im Rechensystem vorstellen. Diese Auffassung beruht auf der Erkenntnis, daß zwischen der Gesellschaft der Menschen und der Gesellschaft von Prozessen eine gewisse Analogie bezüglich der Beurteilung ihrer Leistungsfähigkeit besteht. Bild 1.1 veranschaulicht diese Idee /178/.

Das Schema in Bild 1.1 stützt sich auf zwei Säulen: Die Evolution des Gesellschaftssystems (Rechensystems) und das Verhalten der Menschen (Prozeßverhalten) in der Gesellschaft. Man untersucht einerseits die Struktur (Struktur des Rechensystems), die Organisation (Organisation des Prozeßablaufes) des Gesellschaftssystems mit dessen Verhaltensregeln (z.B. Abfertigungsvorschriften für eine Warteschlange), Gebote und Verbote (z.B. Semaphores) usw., um bessere Modelle des Gesellschaftssystems (Rechnermodelle) bilden zu können. Andererseits benötigt man für eine leistungsfähige Gesellschaft Menschen (Prozesse), die zum Wohlverhalten erzogen sind (z.B.

Prozesse, die keine Verklemmungssituation hervorrufen) oder die sich später angepaßt haben (z.B. Prozesse, deren Verhalten im Adreßraum optimiert worden ist). Um die Gesellschaft dynamisch weiterzuentwickeln und ihre Produktivität (z.B. Durchsatz des Rechensystems) zu steigern, ist es erforderlich, daß immer wieder neue Erkenntnisse und bessere Auffassungen (Denkmodelle, Modellvorstellungen) über die zwischenmenschlichen Beziehungen (gegenseitige Beeinflussung der Prozesse bei Forderung gemeinsamer Betriebsmittel) und über die Wechselwirkungen zwischen Menschen und ihrer Gesellschaft (Wechselwirkung zwischen dem Verhalten der Prozesse und den Kommunikationsmitteln des Rechensystems) gewonnen bzw. aufgestellt werden. Die Voraussetzung dafür ist, daß das Sozialverhalten der Menschen (Prozeßverhalten) und die Geschehnisse in der Gesellschaft (Aktivitäten im Rechensystem) ständig, systematisch und kritisch beobachtet und analysiert werden (Messung des Prozeßverhaltens und Analyse der Meßergebnisse).

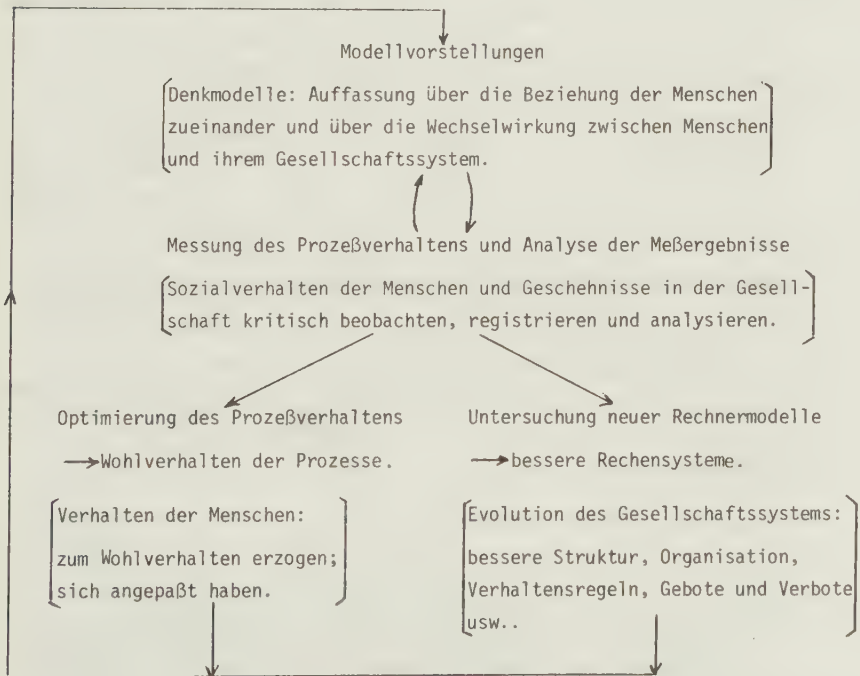


Bild 1.1

Das in Bild 1.1 dargestellte Schema bildet die Grundlage eines Arbeitskonzepts, das sich für die Untersuchung aller mit der Bewertung und mit der Steigerung der Rechnerleistung zusammenhängenden Probleme eignet. Das Arbeitskonzept beruht auf der Modellvorstellung, daß ein Rechensystem im allgemeinen als ein hierarchisch strukturiertes Netz von (sowohl realen als auch virtuellen) Betriebsmitteln zu betrachten ist. Von dieser Modellvorstellung ausgehend lassen sich die nach diesem Konzept durchzuführenden Untersuchungen folgendermaßen klassifizieren:

- i. Messung des Prozeßverhaltens in Rechensystemen.
- ii. Untersuchung von Modellen der Rechensysteme.
- iii. Optimierung des Prozeßverhaltens in Rechensystemen.

1.1.1 Messung des Prozeßverhaltens in Rechensystemen

Die in /175/, Kap. 5, angegebene Messung des dynamischen Programmverhaltens im Adreßraum ist ein exemplarisches Beispiel der speziell auf den Adreßraum /204/ des Rechensystems bezogenen Messungen, wobei das Prozeßverhalten in einem Netz von virtuellen Speicherbereichen /185/ gemessen wurde. Die in /175/, Kapitel 3, beschriebene Messung der Befehlsstatistik und der Rechnerkernphasen läßt sich als Messung des Prozeßverhaltens in einem Netz von virtuellen Funktionseinheiten auffassen, wobei jede Befehlsklasse als eine virtuelle Funktionseinheit betrachtet wird. Unter einer Funktionseinheit wird ein spezieller Prozessor verstanden, der ausschließlich Befehle einer bestimmten Befehlsklasse ausführen kann. Bei den in /196/ durchgeführten Messungen von Programm-Parametern wurde als Rechnermodell ein Netz betrachtet, das aus einem virtuellen Rechnerkern und mehreren virtuellen Ein/Ausgabe-Prozessoren besteht /266/. Andere Messungen (z.B. /180,210/) lassen sich auf gleiche Weise klassifizieren. Eine systematische Analyse der mit Messung und Bewertung zusammenhängenden Probleme ist in /215/ angegeben.

1.1.2 Untersuchung von Modellen der Rechensysteme

Es wird im allgemeinen ein Netz von Betriebsmitteln als Rechnermodell betrachtet. Die Untersuchung verfolgt das Ziel, eine bessere Struktur des Netzes und optimale Organisationsstrategien des Prozeßablaufes im Netz zu bestimmen; dabei lassen sich verschiedene Methoden der Analyse und der Simulation anwenden. Die Analyse und der Entwurf von Strukturen gehören zu den Aufgaben des Rechnerarchitekten /234/; sie bilden die Grundlage für die weiteren Untersuchungen. Um diese Aufgaben zu bewältigen,

werden einerseits umfangreiche Kenntnisse über den neuesten Stand der Hardware- und Software-Technologie vorausgesetzt; andererseits benötigt man einen gewissen Weitblick, um die Entwicklungstendenzen, die Vermarktungsmöglichkeiten und die Nachfrage nach Rechensystemen in der nächsten Zukunft richtig zu erfassen. Die Entscheidung über die endgültige Struktur eines Rechensystems ist i.a. das Ergebnis eines langwierigen und schwierigen Iterationsprozesses, das sowohl von Erfahrungen und Absichten der einzelnen Rechnerarchitekten als auch von Erwägungen über die künftige Organisation des Prozeßablaufes im Rechensystem abhängig ist. Bei jeder Iteration wird von einem Netz mit vorgegebener Struktur ausgegangen. Dabei wird die Frage gestellt, wie der Ablauf der Prozesse im Netz optimal zu organisieren ist. Zu den wichtigsten Gesichtspunkten der Organisation, welche für die Beantwortung dieser Frage relevant sind, gehören:

- a) Strategien der Betriebsmittelverwaltung, die dazu beitragen sollen, bestimmte Leistungskriterien (z.B. Durchsatz, Antwortzeit, Auslastung usw.) des Rechensystems zu optimieren.
- b) Strategien der Betriebsmittelverwaltung, die eine fehlerfreie Zusammenarbeit der asynchron ablaufenden, kooperierenden Prozesse im Netz garantieren.

zu a)

Unter diesem Aspekt wird die Analyse des Ablaufgeschehens i.a. mittels der Theorie der Warteschlangensysteme (z.B. /273/) durchgeführt, wobei rein analytische Verfahren /280/ oder die Methoden der stochastischen Simulation (z.B. /271/) eingesetzt werden können. In beiden Fällen wird das Rechnermodell als Warteschlangennetz betrachtet. Zur Analyse komplexer Warteschlangennetze werden Methoden der Dekomposition /270/ angewendet. In /177/ findet sich ein Überblick über die Entwicklung auf diesem Gebiet in den letzten 10 Jahren.

zu b)

Die fehlerfreie Zusammenarbeit von Prozessen bedeutet, daß im Rechensystem keine Konfliktsituation existiert, sodaß keine Prozesse in den Verklemmungszustand /223,255/ geraten. Für die fehlerfreie Zusammenarbeit gibt es verschiedene Strategien, wie z.B. die Vermeidung von Verklemmungen oder die Erkennung und Beseitigung von Verklemmungen. Man kann auch den Ablauf der Prozesse so organisieren, daß überhaupt keine Verklemmung möglich ist. Zahlreiche Prozeßmodelle sind gebildet worden (z.B. /229,245,247,248/),

um die charakteristischen Merkmale von asynchron ablaufenden Prozessen, wie z.B. Starten, Blockieren und Wecken /254/ von Prozessen, formal zu erfassen und die Verklemmungsprobleme unter den formalen Gesichtspunkten zu untersuchen.

1.1.3 Optimierung des Prozeßverhaltens in Rechensystemen

Man betrachtet hier einen einzelnen Prozeß im Rechensystem und fragt sich, welche Art von Wohlverhalten der Prozeß aufweisen muß, damit eine optimale Leistung des Rechensystems erzielt wird. Zur Beantwortung dieser Frage benötigt man ein Prozeßmodell, das programmorientiert ist. Informell kann man sich das so vorstellen, daß sich der Prozeß zu einem gegebenen Zeitintervall stets auf ein Programm bezieht, dessen dynamischer Ablauf den Prozeß darstellt. Davon ausgehend wird das Prozeßverhalten als eine Folge von Betriebsmittelanforderungen beim dynamischen Ablauf des entsprechenden Programmes aufgefaßt. Dabei läßt sich die Untersuchung unter den folgenden 4 Gesichtspunkten durchführen:

a) Optimierung der Programmlokalität (z.B. /186,188,193,197,202,203,216/)

Gutes Lokalitätsverhalten eines Programmes bedeutet, daß sich die Programm- und Datenzugriffe bei dessen dynamischem Ablauf auf einen bevorzugten, kleinen (virtuellen) Speicherbereich für möglichst lange Zeit konzentrieren; der Übergang von einem bevorzugten Speicherbereich zu einem anderen sollte verhältnismäßig wenig Zeit benötigen. In Kapitel 4 dieser Arbeit werden unter diesem Aspekt Probleme des Prozeßverhaltens im Adreßraum formal erfaßt.

b) Globale Programm-Optimierung (z.B. /145,147,150,156,157/)

Globale Programm-Optimierung verfolgt den Zweck, möglichst viel Rechenzeit und / oder Speicherplatz für die Ausführung des Programms zu sparen. Hierzu gehört z.B. die Verlegung aller bezüglich des Iterationsindizes invarianten arithmetischen Ausdrücke aus den Programmschleifen /148/. Mit der sogenannten "Interval Analysis" (z.B. /144,159/) und der "Flow Graph Analysis" (z.B. /143,152,153,160/) sind auf diesem Gebiet theoretisch fundierte und praktisch brauchbare Algorithmen entwickelt worden (z.B. /151,155,159/). Da die wesentlichste Aufgabe dieser Algorithmen darin besteht, Programmteile, in denen eine Programmschleife enthalten ist, aufzufinden, bilden sie einen guten Ausgangspunkt für die Optimierung der Programmlokalität /176,192/.

c) Fehlerfreie Programme

Zu diesem Gesichtspunkt gehören: Strukturierte Programmierung (z.B. /161,162,163,167,168,172/), Automatische Programmverifikation (z.B. /165,166/) und verschiedene Programm-Testverfahren (z.B. /138/). Der Zweck dieser Untersuchungen besteht darin, fehlerfreie Programme zu erstellen bzw. die Korrektheit der existierenden Programme nachzuweisen. Die Untersuchung der strukturierten Programmierung hat nicht nur zu besseren Einsichten in die statische Ablaufstruktur der Programme und die Datenstrukturen geführt /164,169,170/, sie spielt auch bei der Optimierung der Programmlokalität /173,218/ und bei der globalen Programm-Optimierung /152,153,174/ eine wesentliche Rolle.

d) Optimale Reihenfolge der parallel auszuführenden Programmteile

Die Untersuchung beruht auf der allgemeinen Theorie der Reihenfolgeprobleme /268,269/. Die Voraussetzung dafür ist eine geeignete Methode für die automatische Programm-Analyse /130,137,140/, die alle parallel ausführbaren Anweisungen bzw. Programmabschnitte im Programm auffinden soll. Die parallel auszuführenden Anweisungen und Programmabschnitte lassen sich auch explizit bei der Erstellung des Programmes angeben (z.B. /132/). Für die Beschreibung der parallelen Ausführbarkeit benutzt man graphentheoretische Modelle (z.B. /219,246/). Manche Prozeßmodelle (z.B. /252/) wurden aus solchen graphentheoretischen Modellen abgeleitet.

Diese Arbeit wurde motiviert durch Erkenntnisse, die aus den im Rahmen des Forschungsvorhabens "Adaptive Betriebssysteme" (z.B. /175/) durchgeführten Untersuchungen gewonnen wurden; sie ist als eine Fortsetzung der in /175/, Kapitel 5, berichteten Arbeit "Messung des dynamischen Programmverhaltens im Adreßraum" anzusehen, welche gemäß dem eingangs skizzierten Arbeitskonzept durchgeführt wurde. Es hatte sich aus der Analyse der Meßergebnisse in /175/ u.a. ergeben, daß sich Programme im Adreßraum, der in Bereiche fester Größe eingeteilt ist, sehr schlecht verhalten. D.h. die Ausführungsdauer des einzelnen Programmes in einem virtuellen Speicherbereich ist i.a. so kurz, daß das Programm in dessen dynamischem Ablauf ständig Übergänge von einem virtuellen Speicherbereich zu einem anderen machen muß. Diese Feststellung steht im Gegensatz zur Annahme über das Lokalitätsverhalten der Programme im Adreßraum, auf der fast alle in der Praxis eingesetzten dynamischen Speicherverwaltungsstrategien /183,185,191,195,198,201,205/ beruhen. Für

eine gute Speicherverwaltung braucht man deshalb ein Konzept zur effektiven Optimierung des Lokalitätsverhaltens der Programme. Dies gilt insbesondere für Prozesse, die sich auf häufig auszuführende, im Produktionslauf befindliche Programme beziehen. Bei weiteren Untersuchungen hatte es sich herausgestellt, daß das Kernproblem der Optimierung des Prozeßverhaltens im Adreßraum darin besteht, den entsprechenden Programmgraphen optimal zu zerlegen, wobei ein Programmgraph ein Netz von virtuellen Speicherbereichen bezeichnet. Im nächsten Abschnitt (Abs. 1.2) werden die bisher bekannten Verfahren zur optimalen Zerlegung von Netzen kurz diskutiert; Ideen, die zur Entwicklung der in Kap. 2 und 3 angegebenen neuen Algorithmen geführt haben, werden erläutert. Abschnitt 1.3 enthält allgemeine Bemerkungen über Optimierung des Prozeßverhaltens im Adreßraum; die damit zusammenhängenden Modellvorstellungen werden kurz skizziert.

1.2 Optimale Zerlegung von Netzen - Einführung in die Problematik

Unter einem Netz wird ein endlicher, zusammenhängender, schlichter Graph verstanden, dessen Knoten bzw. dessen Kanten gewichtet sind. Alle graphentheoretischen Grundbegriffe, die für diese Arbeit benötigt werden, sind in Anhang B angegeben. Das Problem der optimalen Zerlegung von Netzen besteht darin, die Knoten des Netzes derart in Blöcken zusammenzufassen, daß die Summe der Knotengewichte in jedem Block eine vorgegebene Gewichtsgrenze nicht übersteigt und daß das Gesamtgewicht der Kanten zwischen den Blöcken minimiert wird. Dieses Problem gehört zum Themenkreis der allgemeinen kombinatorischen Optimierungsaufgaben /53,62,85,88,99,112,275/, das als Modell für viele in der Praxis existierende Probleme anzusehen ist. Solche Probleme sind unter anderem die Segmentierung von Programmen und die Zerlegung von Dateien /64,113,125,127,179,187/, die Zerlegung von elektrischen Netzwerken /121,122/, gewisse Reihenfolgeprobleme /116/, Mustererkennungsprobleme /117/ und soziopolitische Probleme /27/. Die Komplexität des Problems /102/ fällt unter die Klasse der sogenannten "P-Complete"-Probleme /94,104,108/, d.h. der Rechenaufwand für eine exakte Lösung nimmt mit der Mächtigkeit der Knotenmenge des Netzes, welche die "Größe" des Problems kennzeichnet, exponentiell zu. Für die exakte Lösung des Problems, die keiner Einschränkung unterliegt, ist bisher in der Literatur lediglich ein Verfahren bekannt: Das pseudoboolische Programmierungsverfahren von Hammer und Rudeanu /58/. Dabei wird das Problem als eine quadratische Optimierungsaufgabe mit 0 - 1 Variablen /112/ formuliert, die dann mittels des in /58/, Kap. VI, angegebenen Algorithmus zu lösen ist. Das Verfahren hat den Nachteil, daß es sehr aufwendig

ist und sich deshalb nur für die Lösung kleiner Probleme eignet. Für große Probleme nimmt die Anzahl der Rechenschritte bereits bei einem Netz mit 20 oder mehr Knoten so explosionsartig zu, daß sie auch mit Hilfe moderner Rechensysteme nicht ohne sehr großen Aufwand gelöst werden können.

Die bisher in der Praxis zur Lösung großer Probleme vorgeschlagenen Lösungsmethoden sind Annäherungsalgorithmen, die sich folgendermaßen klassifizieren lassen:

- i. Algorithmen, die eine optimale Zerlegung unter vorgegebenen Bedingungen liefern. Solche Bedingungen sind zum Beispiel:
 - a) Alle Kantengewichte müssen eins sein;
 - b) Alle Knoten müssen ein einheitliches Gewicht haben;
 - c) Die Indizes (Knotennummern) der in jedem Block zusammengefaßten Knoten müssen fortlaufende Nummern sein.

Das in /115/ angegebene enumerative Suchverfahren ist ein Beispiel dafür, daß eine Lösung unter der Bedingung a) aufgefunden wird. Unter b) läßt sich das Problem als eine lineare Programmierungsaufgabe mit 0 - 1 Variablen formulieren /64/, die dann mittels des Algorithmus von Hammer und Rudeanu /58/ zu lösen ist. Der Rechenaufwand für die beiden Algorithmen ist leider immer noch zu groß, sodaß sie nicht für brauchbar angesehen werden können. Unter c) existiert bereits ein effizientes Verfahren /118/ zur Lösung des Problems; die Bedingung c) bedeutet jedoch eine Einschränkung, die in ungünstigen Fällen zu einer schlechten Lösung führen kann. Ähnliche Bedingungen bezüglich der Numerierung der Knoten wurden in /124/ vorausgesetzt.

ii. Heuristische Algorithmen

Heuristische Algorithmen sind intelligente Annäherungsverfahren (z.B. /72 - 92/) zur Lösung allgemeiner kombinatorischer Optimierungsaufgaben, welche die spezielle Struktur und die Eigenschaften des zu lösenden Problems ausnutzen, um mit wenig Aufwand die Auffindung einer guten teilloptimalen Lösung zu ermöglichen. Für die optimale Zerlegung von Netzen gibt es zahlreiche Strategien, die allein auf der Analyse der Struktur des Netzes beruhen /113,126,218/; manche andere berücksichtigen dazu die Kantengewichte des Netzes (z.B. /123,124,193,197,202/) oder die Verweildauer auf der Teilmenge von Knoten, die einen Block bilden /127/.

Eine bessere, erfolgversprechendere Strategie besteht darin, das globale stochastische Suchverfahren mit dem lokalen deterministischen Suchverfahren zu kombinieren /88/. Dabei betrachtet man das Verfahren als ein statistisches Experiment /267/ und würfelt bei jedem Versuch um einen Punkt im Lösungsraum, von diesem Punkt aus werden Punkte in dessen unmittelbarer Nachbarschaft gemäß einer deterministischen Vorschrift /89,92/ durchsucht, mit dem Ziel ein lokales Optimum aufzufinden. Die Versuche werden so lange wiederholt, bis die zulässige Rechenzeit die Fortsetzung des Experiments verbietet; das bis dahin erzielte beste lokale Optimum wird als Lösung des Problems betrachtet. Auf diese Weise kann mit wenig Aufwand eine gute approximierende Lösung erzielt werden (z.B. /82,84,85,86,119/). Das Verfahren ist leider nicht in der Lage, exakte Angabe über die Fehlerschranken der Approximation zu liefern. Heuristische Verfahren, die eine approximierende Lösung mit garantierten Fehlerschranken liefern, sind Gegenstand zahlreiche Untersuchungen (z.B. /73,76,77,90,91/), die in den letzten Jahren durchgeführt wurden. Eine andere Möglichkeit, Approximation mit sehr wenig Aufwand durchzuführen, besteht darin, die oberen und die unteren Grenzwerte der Lösung zu berechnen (z.B. /80,81,97,101, 103,107/).

In Kapitel 2 dieser Arbeit wird ein mathematisch begründetes, praktisch brauchbares neues Rechenverfahren zur optimalen Zerlegung von Netzen entwickelt. Das Rechenverfahren ermöglicht es, die in der praxis auftretenden Probleme mit vertretbarem Rechenaufwand exakt zu lösen; es ist darüberhinaus in der Lage, für sehr große Probleme eine approximierende Lösung mit vorgegebenen bzw. mit exakt berechneten Fehlerschranken zu liefern. Das Rechenverfahren beruht auf einem Konzept, das darauf abzielt, das eigentliche Problem der optimalen Zerlegung von Netzen in mehrere, einfacher zu lösende Teilprobleme zu zerlegen, welche dann nacheinander gelöst werden. Die Lösungen eines Teilproblems werden als gegebene Parameter des darauffolgenden Teilproblems benutzt. Die Lösung des eigentlichen Problems ergibt sich dann aus der Lösung des letzten Teilproblems.

In Abschnitt 2.1 wird das Problem der optimalen Zerlegung von Netzen definiert. Die Idee, die zur Einteilung des Problems führt, wird in Abschnitt 2.2 erläutert. Es hat sich herausgestellt, daß das erste Teilproblem mit rein kombinatorischem Verfahren gelöst werden kann. Dieses Verfahren wird in Abschnitt 2.3 angegeben; es hat jedoch den Nachteil, daß sich in ungünstigen Fällen eine riesige Menge von Lösungen des Teilproblems ergibt, in der Elemente enthalten sind, die zum größten Teil für die Lösung des eigentlichen Problems

keine Bedeutung haben. Der Grund liegt darin, daß bei der rein kombinatorischen Methode die Strukturen der Teilgraphen in den sich aus der Zerlegung ergebenden Blöcken nicht berücksichtigt werden; für die Bestimmung des Optimums sind jedoch ausschließlich diejenigen Blöcke von Belang, deren Teilgraphen zusammenhängend sind. Diese Erkenntnis hat entscheidend zur Vereinfachung des Rechenverfahrens und zur Reduzierung seiner Komplexität beigetragen; damit ist die praktische Brauchbarkeit des Konzepts gesichert. Das Konzept besteht aus 3 nacheinander durchzuführende Rechenphasen; jeder Rechenphase wird ein Prozeß zugeordnet, der das entsprechende Teilproblem löst. Diese Prozesse sind: Der Erweiterungsprozeß, der binäre Verzweigungsprozeß und der Ergänzungsprozeß. In den Abschnitten 2.4, 2.5 und 2.6 werden diese 3 Prozesse formal erfaßt; ihre Eigenschaften werden begründet. Eine zusammenfassende Darstellung des Konzepts wird in Abschnitt 2.7 angegeben. Die wesentlichen Rechenschritte dieser Prozesse bestehen aus Suchoperationen und logischen Verknüpfungen von boolischen Vektoren. Deshalb eignen sie sich besonders gut für die Implementierung in einer Rechenanlage. In Abschnitt 2.8 werden schließlich mehrere an verschiedenen Literaturstellen angegebene Probleme (Netze mit 14- 13 Knoten und beliebiger Knoten und Kantengewichtung) als Beispiel mittels dieses neuen Rechenverfahrens manuell, d.h. mit Bleistift und Papier allein, gelöst. Das manuelle Rechnen und die damit zusammenhängende schematische Darstellung der Prozesse sollen dazu beitragen, bessere Einsichten in die Arbeitsweise der einzelnen Prozesse zu geben.

Verschiedene Möglichkeiten zur Approximation mittels dieses neuen Rechenverfahrens werden in Kapitel 3 angegeben und diskutiert. Es hat sich herausgestellt, daß das Rechenverfahren bei der Lösung komplexer Probleme, bei denen die vorgegebene Rechenzeit bzw. Speicherkapazität nicht überschritten werden darf, bezüglich der Wahl der Annäherungsmethoden eine sehr große Flexibilität aufweist. Die Approximation läßt sich bei der ersten oder bei der zweiten Phase des Konzepts durchführen, wobei sie je nach der angewandten Methode eine Lösung mit vorgegebenem relativen Fehler (Abs. 3.2), eine Lösung mit möglichst kleinem relativen Fehler (Abs. 3.3) oder eine Lösung mit exakt berechneten Fehlerschranken (Abs. 3.4) liefert. Eine exakte Lösung, die man unter zusätzlichen Bedingungen mit wenig Rechenaufwand erhält, läßt sich als eine Approximation betrachten. Diese Art von Approximation wird in Abschnitt 3.5 diskutiert.

1.3 Allgemeine Bemerkungen zur Optimierung des Prozeßverhaltens im Adreßraum

Die "Optimierung des Prozeßverhaltens im Adreßraum" wird oft im Zusammenhang mit den Problemen der statischen Speicherverwaltung "Segmentierung von Programmen" genannt (z.B. /175/). An verschiedenen englischsprachigen Literaturstellen erhält sie unterschiedliche Bezeichnungen: "Programm-Restructuring" /188,193/, "Prepaging" /216/, "Paging" /24,185,192,218/, "Program segmentation" bzw. "Record segmentation" /64,113,187,197,202/ usw.. Mit wenigen Ausnahmen (z.B. /187,202,216/) beschäftigen sich die eben erwähnten Arbeiten mit der optimalen Zerlegung von Programmen allein, ohne die dazu gehörigen Datenmengen zu berücksichtigen. Mit dem ständig zunehmenden Bedarf an Datenbanken gewinnt das Problem der Verwaltung großer strukturierter Datenmengen /263/ in Rechensystemen immer mehr an Bedeutung. Untersuchungen, die sich ausschließlich auf die Zerlegung von strukturierten Datenmengen bzw. Dateien beziehen, sind z.B. in /187,216/ angegeben. Andere relevante Untersuchungen (z.B. /194,207,208/) wurden im Zusammenhang mit den Methoden der Dateiorganisation /182,199,200,206,212,213/ und mit den Problemen der dynamisch zu erweiternden Datenstrukturen /208/ durchgeführt.

Für die Untersuchung des Prozeßverhaltens im Adreßraum - kurz, des Prozeßverhaltens - werden in Kapitel 4 dieser Arbeit Programme und ihre Datenmengen gleichzeitig behandelt. Dabei wird vorausgesetzt, daß der Adreßraum des Rechensystems segmentiert und jedes Segment in zusammenhängende Bereiche fester Größe, d.h. Seiten, eingeteilt ist /204/. Die Segmente und die Seiten werden als einheitliche virtuelle Speicherbereiche betrachtet, die zur Vereinfachung der Speicherverwaltung beitragen sollen. Für die Betrachtung des Prozeßverhaltens benötigt man ein Prozeßmodell, das programmorientiert ist. D.h. ein Prozeß wird als dynamischer Ablauf des Programmes aufgefaßt, auf das er sich bezieht. Diese Auffassung stimmt im Grunde mit der in /175/, Kap. 5, aufgestellten Modellvorstellung überein, wonach das dynamische Programmverhalten im Adreßraum analog zum Verhalten einer futter-suchenden Maus (Prozeß) im Labyrinth (Adreßraum des entsprechenden Programmes) betrachtet wird. Um die auf den Adreßraum bezogene Struktur des Programmes anzugeben, wird der speicherorientierte Programmgraph eingeführt (Abs. 4.2), der die Ablaufstruktur des Programmes im Adreßraum darstellt. Entsprechend der Detailliertheit, mit der die Struktur des Programmes durch den Programmgraphen dargestellt wird, kann das Prozeßverhalten beschrieben werden. Um das Prozeßverhalten möglichst architekturneutral /178/ bzw. unabhängig von etwaigen Programmiersprachen anzugeben, wird die Sprache von Luckham, Park und Paterson /171/ benutzt. Damit lassen sich Programme und Daten formal

darstellen. Solche formalen uninterpretierten Programme werden Programmschemata genannt; dementsprechend wird die Sprache als Schematasprache bezeichnet. Sie ist als Modell der internen Sprachen /144/ von Rechensystemen anzusehen. Von diesem Modell ausgehend werden die Begriffe "linearer Programmabschnitt" und "Datenblock" entwickelt". Der lineare Programmabschnitt bildet den Programmknoten, der Datenblock den Datenknoten des speicherorientierten Programmgraphen. Der Übergang von einem Programmknoten zu einem anderen wird durch eine gerichtete Kante dargestellt; eine ungerichtete Kante zwischen einem Programmknoten und einem Datenknoten kennzeichnet den möglichen Datenzugriff, der vom Programmknoten aus durchgeführt wird. Alle elementaren Begriffe, die für die formale Darlegung der in den Abschnitten 4.2, 4.3 und 4.4 zu behandelnden Probleme nötig sind, werden in Abschnitt 4.1 eingeführt.

Auf dem programmorientierten Prozeßmodell beruhend wird in Abschnitt 4.3 der ganze Ablauf des Prozesses als eine Tour durch den Adreßraum desjenigen Programmes aufgefaßt, auf das sich der Prozeß bezieht. Eine Tour besteht aus einer nicht leeren, endlichen Folge von Zuständen; der Prozeß verweilt in jedem Zustand für ein bestimmtes Zeitintervall, wobei verschiedene Aktivitäten, wie z.B. Anweisungen ausführen, Daten zugreifen usw., ausgelöst werden. Informell wird die Gesamtheit der Aktivitäten in einem Zustand als eine Aktivitätsphase des Prozesses betrachtet. Das Prozeßverhalten wird also beschrieben als eine nicht leere, endliche Folge von Aktivitätsphasen, in denen Informationen (z.B. Häufigkeit der Datenzugriffe in einem Zustand) enthalten sind, die zur Bestimmung der Parameter des Programmgraphen zur Verfügung gestellt werden können. Es wird nachgewiesen, daß die Optimierung des Prozeßverhaltens identisch ist mit der optimalen Zerlegung von speicherorientierten Programmgraphen.

Zum Schluß werden in Abschnitt 4.4 praktische Überlegungen zur Optimierung des Prozeßverhaltens diskutiert. Es wird insbesondere auf die Schwierigkeiten zur Durchführung der statischen und dynamischen Messungen /175/ hingewiesen, die für die Auffindung der speicherorientierten Programmgraphen unentbehrlich sind. Unter der Voraussetzung, daß der Adreßraum segmentiert und jedes Segment in Seiten eingeteilt ist, wird für den speicherorientierten Programmgraphen eine zweistufige Zerlegungsverfahren vorgeschlagen, wobei jede Stufe aus einer optimalen Zerlegung besteht, die mittels des in Kapitel 2 entwickelten Rechenverfahrens durchgeführt wird.

2. Die optimale Zerlegung von Netzen - exakte Lösung

2.1 Das Problem

Sei V eine endliche Menge, die Knotenmenge genannt wird.

Definition 2.1.1

Ein endlicher, gerichteter Graph ist ein Paar $G_d = (V, A)$, wobei

$$A \subseteq V \times V \setminus \Delta_V .$$

Ein endlicher, ungerichteter Graph ist ein Paar $G_u = (V, E)$, wobei

$$E \subseteq \binom{V}{2} .$$

A und E heißen Kantenmenge. Ein Element $(a,b) \in A$ ($\{a,b\} \in E$) heißt gerichtete Kante (ungerichtete Kante).

Bemerkung 2.1.2

G_d und G_u gehören zu einer Klasse von Graphen, die weder Mehrfachkanten noch Schlingen besitzen. Graphen mit dieser Eigenschaft werden im allgemeinen als schlichte Graphen bezeichnet. In dieser Arbeit werden wir ausschließlich endliche, schlichte Graphen betrachten. Die wichtigsten Grundbegriffe und Eigenschaften der schlichten Graphen werden in Anhang B eingeführt; sie beschränken sich auf Definitionen und deren unmittelbare Folgerungen, die im Zusammenhang mit den zu behandelnden Themen eine Rolle spielen. Für eine ausführliche Behandlung der allgemeinen Graphentheorie siehe z.B. /262,265,277/. In Anhang A findet sich eine Zusammenfassung der in dieser Arbeit benutzten allgemeinen Notationen. Andere spezielle Bezeichnungen werden im Bedarfsfall an geeigneten Stellen dieser Arbeit angegeben.

Definition 2.1.3

Ein ungerichtetes (gerichtetes) Netz ist ein Quadrupel

$$NG = (V, E, g, f) \quad (ND = (V, A, g, f')),$$

wobei folgendes gilt:

- i) (V, E) ((V, A)) ist ein endlicher, ungerichteter (gerichteter) Graph.
- ii) (V, E) ((V, A)) ist zusammenhängend.

iii) $g : V \longrightarrow \mathbb{N}$ ist eine Abbildung, die Knotengewichtung genannt wird.

iv) $f : E \longrightarrow \mathbb{N}$ ($f' : A \longrightarrow \mathbb{N}$) ist eine Abbildung, die Kantengewichtung genannt wird.

Sei $a \in V$, dann heit $g(a)$ Knotengewicht (kurz, Gewicht) von a . Sei $\{a,b\} \in E$ ($(a,b) \in A$), dann heit $f(\{a,b\})$ ($f'((a,b))$) Kantengewicht (kurz, Gewicht) der Kante $\{a,b\}$ ((a,b)).

Bemerkung 2.1.4

Sei $|V| = n$. Man bentigt fr jeden der n Knoten in V eine Bezeichnung. Anstatt wie blich zu diesem Zweck die Knotenmenge mit Hilfe der Indexmenge $\mathbb{N}_n$ zu indizieren, werden wir der Einfachheit wegen annehmen, da $V = \mathbb{N}_n$. Diese Annahme bedeutet fr die beabsichtigten Untersuchungen keine Einschrnkung der Allgemeinheit. Von dieser Annahme ausgehend werden wir zweckmigerweise statt Abbildungen die Indeschreibweise fr g und f (bzw. f') benutzen:

$g \hat{=} (g_i \mid i \in \mathbb{N}_n)$ D.h. g ist eine endliche Folge.

$f \hat{=} (f_{ij} \mid i, j \in \mathbb{N}_n \wedge f_{ij} = f_{ji})$ und

$f' \hat{=} (f'_{ij} \mid i, j \in \mathbb{N}_n)$.

D.h. f (bzw. f') ist eine $n \times n$ Matrix, fr die folgendes gilt:

$$f_{ij} \text{ (f'_{ij})} \begin{cases} \in \mathbb{N}, \text{ wenn } \{i, j\} \in E \text{ (} (i, j) \in A \text{)} \\ = 0, \text{ sonst.} \end{cases}$$

Definition 2.1.5

Sei $ND = (V', A, g', f')$ ein gerichtetes Netz. $NG = (V, E, g, f)$ ist genau dann das abgeleitete Netz von ND , wenn gilt:

i) $V = V'$

ii) $g = g'$

iii) $\forall i \forall j ((i, j) \in A \vee (j, i) \in A \Rightarrow \{i, j\} \in E)$

iv) $\forall i \forall j (f_{ij} = f_{ji} = f'_{ij} + f'_{ji})$

Definition 2.1.6

Sei $NG = (V, E, g, f)$ ($ND = (V, A, g, f')$) ein ungerichtetes (gerichtetes) Netz. Sei $a \in \mathbb{N}$, und es gelte $\sum_{i \in V} g_i \leq a$ und $a \geq g_i$ für $\forall i \in V$.

i) $B \subset 2^V$ heißt Zerlegung von V genau dann, wenn gilt:

$$\emptyset \notin B$$

$$\cup B = V$$

$$\forall b, b' \in B (b \neq b' \Rightarrow b \cap b' = \emptyset)$$

ii) Eine Zerlegung heißt bezüglich (g, a) zulässig (kurz, zulässig) genau dann, wenn

$$\sum_{j \in b} g_j \leq a \quad \text{für } \forall b \in B.$$

a heißt Gewichtsgrenze. Ist B eine zulässige Zerlegung, dann heißt $b \in B$ Block.

Die Menge aller zulässigen Zerlegungen von V wird mit π bezeichnet.

iii) Sei b ein Block. Dann heißt (b, E_b) mit $E_b = \binom{b}{2} \cap E$ ((b, A_b) mit $A_b = (b \times b) \cap A$) dem Block b zugeordneter Teilgraph (kurz, Blockteilgraph).

iv) Die Abbildung $w : B \rightarrow \mathbb{N}_0$, angegeben durch

$$w(b) = \frac{1}{2} \sum_{\{k, l\} \in E_b} f_{kl}$$

$$(w(b) = \sum_{(k, l) \in A_b} f'_{kl})$$

wird Blockbewertung genannt. $w(b)$ heißt Wert des Blockes b .

v) Die Abbildung $Z : \pi \rightarrow \mathbb{N}_0$ heißt Zielfunktion, für die gilt:

$$Z(B) = \sum_{b \in B} w(b).$$

$Z(B)$ heißt Wert von B . Eine zulässige Zerlegung $\hat{B} \in \pi$ heißt optimal genau dann, wenn $Z(\hat{B}) = \max \{Z(B) \mid B \in \pi\}$.

$\hat{Z} = Z(\hat{B})$ heißt Optimum.

vi) $P = (NG, a, Z)$ ($P' = (ND, a, Z)$) wird Problem der optimalen Zerlegung von NG (ND) (kurz, Problem) genannt.

Sei $\hat{\pi} = \{B \mid B \in \pi \wedge Z(B) = \hat{Z}\}$ die Menge der optimalen Zerlegungen.

$L = \hat{\pi} \times \{\hat{Z}\}$ wird Lösungsmenge des Problems genannt.

Jedes $(\hat{B}, \hat{Z}) \in L$ ist eine Lösung des Problems.

Definition 2.1.7

Sei Q eine Menge von Problemen. Seien $P_1, P_2 \in Q$. Sei L_1 (L_2) die Lösungsmenge von P_1 (P_2).

P_1 und P_2 heißen äquivalent (kurz, $P_1 \cong P_2$) g.d.w. $L_1 = L_2$.

P_1 und P_2 heißen schwach äquivalent (kurz, $P_1 \sim P_2$) g.d.w. für $\forall (\hat{B}_1, \hat{Z}_1) \in L_1$

und $\forall (\hat{B}_2, \hat{Z}_2) \in L_2$ folgendes gilt:

$$\hat{Z}_1 = \hat{Z}_2 \wedge (\hat{B}_1 \subseteq \hat{B}_2 \vee \hat{B}_2 \subseteq \hat{B}_1).$$

Folgerung 2.1.8

i) $\cong \subseteq Q \times Q$ ist eine Äquivalenzrelation auf Q .

ii) $P_1 \cong P_2 \implies P_1 \sim P_2$.

Satz 2.1.9

Sei $NG = (V, E, g, f)$ ein ungerichtetes Netz. Sei $ND = (V', A, g', f')$ ein gerichtetes Netz. Seien $P = (NG, a, Z)$ und $P' = (ND, a', Z')$ Probleme.

NG ist das abgeleitete Netz von ND $\wedge a = a' \implies P \cong P'$.

Beweis:

Sei π (π') die Menge aller zulässigen Zerlegungen von V (V').

Sei $\hat{\pi}$ ($\hat{\pi}'$) die Menge der optimalen Zerlegungen. Sei L (L') die Lösungsmenge von P (P').

Aus $a = a'$ und den Bedingungen i) und ii) der Definition 2.1.5 folgt offensichtlich, daß $\pi = \pi'$.

Seien $B \in \pi$, $B' \in \pi'$. Es muß zunächst nachgewiesen werden, daß

$$B = B' \implies Z(B) = Z'(B').$$

$$\begin{aligned}
Z(B) &= \frac{1}{2} \sum_{b \in B} \sum_{\{k, l\} \in E} f_{kl} && (\text{Def. 2.1.6 iv, v}) \\
&= \sum_{b \in B} \sum_{\substack{\{k, l\} \in E \\ k < l}} f_{kl} \\
&= \sum_{b \in B'} \left(\sum_{\substack{(k, l) \in A \\ k < l}} f'_{kl} + \sum_{\substack{(l, k) \in A \\ k < l}} f'_{lk} \right) && (B = B' \text{ und Def. 2.1.5 iv}) \\
&= \sum_{b \in B'} \sum_{(k, l) \in A} f'_{kl} \\
&= Z'(B'). && (\text{Def. 2.1.6 iv, v})
\end{aligned}$$

Sei $\hat{Z}$ ($\hat{Z}'$) das Optimum und $\hat{\pi}$ ($\hat{\pi}'$) die Menge der optimalen Zerlegung von P (P').

$$\begin{aligned}
\hat{Z} &= \text{Max} \{ Z(B) \mid B \in \pi \} && (\text{Def. 2.1.6 vi}) \\
&= \text{Max} \{ Z'(B') \mid B' \in \pi' \} && (\pi = \pi' \wedge B = B' \Rightarrow Z(B) = Z'(B')) \\
&= \hat{Z}'.
\end{aligned}$$

Daraus folgt, daß $\hat{\pi} = \hat{\pi}'$ und $L = L'$. $\square$

Bemerkung 2.1.10

Satz 2.1.9 sagt aus, daß man die gleiche Lösungsmenge erhält, wenn anstelle eines gegebenen gerichteten Netzes dessen abgeleitetes Netz zerlegt wird. Im Vergleich zum gerichteten Netz benötigt das abgeleitete Netz etwa die Hälfte des Speicherplatzes im Rechensystem zur Speicherung der Kantenmenge und der Kantengewichtung. Aus diesem Grunde werden wir uns künftig ausschließlich mit den Problemen der optimalen Zerlegung von ungerichteten Netzen beschäftigen.

Im folgenden sei $NG = (V, E, g, f)$ ein ungerichtetes Netz und $P = (NG, a, Z)$ ein Problem. Sei $B = \{b_i \mid i \in \mathbb{N}_m\}$ mit $|B| = m$ eine zulässige Zerlegung von V .

Definition 2.1.11

Ein durch B induziertes Netz ist ein Quadrupel $NZ_B = (B, K, q, h)$, wobei für B, K, q und h folgendes gilt:

B ist die Knotenmenge.

$K \subseteq \binom{B}{2}$ ist die Kantenmenge, angegeben durch

$$\forall b_i, \forall b_j (\{b_i, b_j\} \in K \iff \exists u \exists v (u \in b_i \wedge v \in b_j \Rightarrow \{u, v\} \in E)).$$

$q : B \longrightarrow \mathbb{N}$ ist die Knotengewichtung, für die gilt:

$$q(b) = \sum_{1 \in b} g_1 \quad \text{für } \forall b \in B.$$

$h : K \longrightarrow \mathbb{N}$ ist die Kantengewichtung, wobei

$$h(\{b_i, b_j\}) = \sum_{\substack{u \in b_i \\ v \in b_j \\ \{u, v\} \in E}} f_{uv} \quad \text{die Übergangshäufigkeit zwischen } b_i \text{ und } b_j \text{ genannt wird.}$$

Ist B optimal, dann heißt $NZ_B = (B, K, q, h)$ optimale Zerlegung von NG.

Definition 2.1.12

Sei $b_i \in B$ ein Block.

Die Abbildung $x_i : V \longrightarrow \{0, 1\}$, angegeben durch

$$x_i(j) \equiv \begin{cases} 1, & \text{wenn } j \in b_i \\ 0, & \text{wenn } j \in V \setminus b_i, \end{cases}$$

heißt charakteristische Funktion von b_i bezüglich V .

Bezeichnung 2.1.13

x_i ist ein Element aus dem n -dimensionalen Vektorraum über $GF(2)$. Statt $x_i(j)$ wird künftig x_{ij} geschrieben. Mit $\underline{X}_i = (x_{i1}, \dots, x_{in}) \in \{0, 1\}^n$ wird der charakteristische Vektor von b_i bezeichnet, dabei ist x_{ij} die j -te Komponente von $\underline{X}_i$. Wenn $x_{ij} = 0$ (bzw. $x_{ij} = 1$), dann wird x_{ij} als Null-Komponente (bzw. Eins-Komponente) von $\underline{X}_i$ bezeichnet. I_i bezeichnet die Menge der Indizes für die Eins-Komponenten von $\underline{X}_i$, d.h. $I_i \equiv \{j \mid j \in \mathbb{N}_n \wedge x_{ij} = 1\}$. $\underline{0}$ bezeichnet den Null-Vektor $(0, \dots, 0)$, $\underline{1}$ den Eins-Vektor $(1, \dots, 1)$, und $\underline{e}_j$ den Einheitsvektor $(0, \dots, 0, 1, 0, \dots, 0)$. Die j -te Komponente von $\underline{e}_j$ ist seine einzige Eins-Komponente.

Seien $\vee$ oder, $\wedge$ = und, $\oplus$ = modulo 2 Addition und $+$ = arithmetische Addition. Seien $k, l \in \mathbb{N}_m$.

$$\bigvee_{i \in \mathbb{N}_m} \underline{X}_i = \underline{X}_1 \vee \dots \vee \underline{X}_m, \text{ wobei } \underline{X}_k \vee \underline{X}_l = (x_{k1} \vee x_{l1}, \dots, x_{kn} \vee x_{ln}).$$

$$\bigwedge_{i \in \mathbb{N}_m} \underline{X}_i = \underline{X}_1 \wedge \dots \wedge \underline{X}_m, \text{ wobei } \underline{X}_k \wedge \underline{X}_l = (x_{k1} \wedge x_{l1}, \dots, x_{kn} \wedge x_{ln}).$$

$$\bigoplus_{i \in \mathbb{N}_m} \underline{X}_i = \underline{X}_1 \oplus \dots \oplus \underline{X}_m, \text{ wobei } \underline{X}_k \oplus \underline{X}_l = (x_{k1} \oplus x_{l1}, \dots, x_{kn} \oplus x_{ln}).$$

$$\sum_{i \in \mathbb{N}_m} \underline{X}_i = \underline{X}_1 + \dots + \underline{X}_m, \text{ wobei } \underline{X}_k + \underline{X}_l = (x_{k1} + x_{l1}, \dots, x_{kn} + x_{ln}).$$

Lemma 2.1.14

$$\sum_{j \in b_i} g_j \leq a \iff \sum_{j \in \mathbb{N}_n} g_j x_{ij} \leq a \quad \forall i \in \mathbb{N}_m$$

Beweis:

Aus der Definition 2.1.12 folgt, daß $\sum_{j \in b_i} g_j = \sum_{j \in \mathbb{N}_n} g_j x_{ij} \cdot \square$

Lemma 2.1.15

Sei $b_i \in B$ und (b_i, E_i) ein Blockteilgraph.

- i) $\{k, l\} \in E_i \Rightarrow x_{ik} x_{il} = 1.$
- ii) $k \notin b_i \vee l \notin b_i \Rightarrow x_{ik} x_{il} = 0.$
- iii) $k \in b_i \wedge l \notin b_i \Rightarrow x_{ik}(1 - x_{il}) = 1.$

Beweis: Offensichtlich. $\square$

Lemma 2.1.16

- i) $i \in \mathbb{N}_m (b_i \neq \emptyset) \iff \forall i \in \mathbb{N}_m (\underline{x}_i \neq \underline{0}).$
- ii) $\bigcup_{i \in \mathbb{N}_m} b_i = V \iff \bigvee_{i \in \mathbb{N}_m} \underline{x}_i = \underline{1}.$
- iii) $\forall i, j \in \mathbb{N}_m (i \neq j \Rightarrow b_i \cap b_j = \emptyset) \iff \forall i, j \in \mathbb{N}_m (i \neq j \Rightarrow \underline{x}_i \wedge \underline{x}_j = \underline{0}).$

Beweis: Offensichtlich. $\square$

Definition 2.1.17

Sei $\underline{x}_i$ der charakteristische Vektor von $b_i \in B$. Mit $U = \{\underline{x}_i \mid i \in \mathbb{N}_m\}$ wird die Menge der charakteristischen Vektoren von B bezeichnet.

$$v(\underline{x}_i) \hat{=} \frac{1}{2} \sum_k \sum_{l \in \mathbb{N}_n} f_{kl} x_{ik} x_{il} \quad \text{heißt } \underline{\text{Wert}} \text{ von } \underline{x}_i$$

$$W(U) \hat{=} \sum_{i \in \mathbb{N}_m} v(\underline{x}_i) \quad \text{heißt } \underline{\text{Wert}} \text{ von } U.$$

Lemma 2.1.18

$$w(b_i) = v(X_i)$$

$$\text{Beweis:} \quad w(b_i) = \frac{1}{2} \sum_{\{k,l\} \in E_i} f_{kl} \quad (\text{Def. 2.1.6 iv})$$

$$= \frac{1}{2} \sum_k \sum_{\substack{l \\ k,l \in \mathbb{N}_n}} f_{kl} x_{ik} x_{il} \quad (\text{Lemma 2.1.15 i})$$

$$= v(X_i). \quad \square$$

Lemma 2.1.19

$$Z(B) = W(U) = \frac{1}{2} \sum_{i \in \mathbb{N}_m} \sum_k \sum_{\substack{l \\ k,l \in \mathbb{N}_n}} f_{kl} x_{ik} x_{il}.$$

$$\text{Beweis:} \quad Z(B) = \sum_{b_i \in B} w(b_i) \quad (\text{Def. 2.1.6 v})$$

$$= \sum_{i \in \mathbb{N}_m} v(X_i) \quad (\text{Lemma 2.1.18})$$

$$= W(U) \quad (\text{Def. 2.1.17})$$

$$= \frac{1}{2} \sum_{i \in \mathbb{N}_m} \sum_k \sum_{\substack{l \\ k,l \in \mathbb{N}_n}} f_{kl} x_{ik} x_{il}. \quad \square$$

Statt $Z(B)$ bzw. statt $W(U)$ werden wir künftig Z als Bezeichnung der Zielfunktion des Problems schreiben. Es gilt also:

$$Z = \frac{1}{2} \sum_{i \in \mathbb{N}_m} \sum_k \sum_{\substack{l \\ k,l \in \mathbb{N}_n}} f_{kl} x_{ik} x_{il}.$$

Mit $\hat{Z}$ wird das Maximum der Zielfunktion bezeichnet.

Lemma 2.1.20

$$\forall i, j \in \mathbb{N}_m (i \neq j \Rightarrow \underline{x}_i \wedge \underline{x}_j = \underline{0}) \wedge (\bigvee_{i \in \mathbb{N}_m} \underline{x}_i = \underline{1}) \Leftrightarrow \sum_{i \in \mathbb{N}_m} \underline{x}_i = \underline{1}$$

Beweis:

" $\Rightarrow$ "

$$\text{Annahme: } (i \neq j \Rightarrow \underline{x}_i \wedge \underline{x}_j = \underline{0}) \wedge (\bigvee_{i \in \mathbb{N}_m} \underline{x}_i = \underline{1}) \Rightarrow \sum_{i \in \mathbb{N}_m} \underline{x}_i \neq \underline{1}$$

Für $\sum_{i \in \mathbb{N}_m} \underline{x}_i \neq \underline{1}$ gibt es zwei Möglichkeiten:

$$\text{i) } \exists k \in \mathbb{N}_n (\sum_{i \in \mathbb{N}_m} x_{ik} = 0)$$

Dies bedeutet, daß $\exists k (\bigvee_{i \in \mathbb{N}_m} x_{ik} = 0) \Rightarrow \bigvee_{i \in \mathbb{N}_m} \underline{x}_i \neq \underline{1}$. Widerspruch.

$$\text{ii) } \exists k \in \mathbb{N}_n (\sum_{i \in \mathbb{N}_m} x_{ik} > 1)$$

Dies bedeutet, daß $\exists i \exists j \exists k (i \neq j \Rightarrow x_{ik} \wedge x_{jk} = 1 \Rightarrow \underline{x}_i \wedge \underline{x}_j \neq \underline{0})$.

Widerspruch.

" $\Leftarrow$ "

$$\text{Annahme: } \sum_{i \in \mathbb{N}_m} \underline{x}_i = \underline{1} \Rightarrow \exists i \exists j (i \neq j \Rightarrow \underline{x}_i \wedge \underline{x}_j \neq \underline{0}) \vee (\bigvee_{i \in \mathbb{N}_m} \underline{x}_i \neq \underline{1})$$

$$\text{Fall 1: } \exists i \exists j (i \neq j \Rightarrow \underline{x}_i \wedge \underline{x}_j \neq \underline{0}) \Rightarrow \exists i \exists j \exists k (x_{ik} \wedge x_{jk} \neq 0)$$

$$\Rightarrow \exists i \exists j \exists k (x_{ik} \wedge x_{jk} = 1)$$

$$\Rightarrow \exists i \exists j \exists k (x_{ik} + x_{jk} > 1)$$

$$\Rightarrow \sum_{i \in \mathbb{N}_m} \underline{x}_i \neq \underline{1}. \text{ Widerspruch.}$$

$$\text{Fall 2: } \bigvee_{i \in \mathbb{N}_m} \underline{x}_i \neq \underline{1} \Rightarrow \bigvee_{i \in \mathbb{N}_m} \underline{x}_i = \underline{0}$$

$$\Rightarrow \forall i \in \mathbb{N}_m (\underline{x}_i = \underline{0})$$

$$\Rightarrow \sum_{i \in \mathbb{N}_m} \underline{x}_i = \underline{0}. \text{ Widerspruch. } \square$$

Satz 2.1.21

$B = \{b_i \mid i \in \mathbb{N}_m\}$ ist eine zulässige Zerlegung g.d.w. gilt:

$$i) \forall i \in \mathbb{N}_m \left(\sum_{j \in \mathbb{N}_n} x_{ij} \neq 0 \right)$$

$$ii) \forall j \in \mathbb{N}_n \left(\sum_{i \in \mathbb{N}_m} x_{ij} = 1 \right)$$

$$iii) \forall i \in \mathbb{N}_m \left(\sum_{j \in \mathbb{N}_n} g_j x_{ij} \leq a \right)$$

Beweis:

Nach Def. 2.1.6 i und ii) ist B genau dann eine zulässige Zerlegung wenn gilt:

$$1) \forall i \in \mathbb{N}_m (b_i \neq \emptyset)$$

$$2) \bigcup_{i \in \mathbb{N}_m} b_i = V$$

$$3) \forall i, j \in \mathbb{N}_m (i \neq j \Rightarrow b_i \cap b_j = \emptyset)$$

$$4) \sum_{j \in b_i} g_j \leq a \quad \text{für } \forall i \in \mathbb{N}_m.$$

Für die Bedingungen 1) - 4) betrachtet man nun statt Mengen ihre Charakteristischen Vektoren. Die Aussagen 1') - 3') sind identisch mit dem Lemma 2.1.16.

$$1') \forall i \in \mathbb{N}_m (b_i \neq \emptyset) \Leftrightarrow \forall i \in \mathbb{N}_m (\underline{x}_i \neq \underline{0}) \Leftrightarrow \forall i \in \mathbb{N}_m \left(\sum_{j \in \mathbb{N}_n} x_{ij} \neq 0 \right)$$

$$2') \bigcup_{i \in \mathbb{N}_m} b_i = V \Leftrightarrow \bigvee_{i \in \mathbb{N}_m} \underline{x}_i = \underline{1}$$

$$3') \forall i, j \in \mathbb{N}_m (i \neq j \Rightarrow b_i \cap b_j = \emptyset) \Leftrightarrow \forall i, j \in \mathbb{N}_m (i \neq j \Rightarrow \underline{x}_i \wedge \underline{x}_j = \underline{0})$$

$$4') \forall i \in \mathbb{N}_m \left(\sum_{j \in b_i} g_j \leq a \Leftrightarrow \sum_{j \in \mathbb{N}_n} g_j x_{ij} \leq a \right) \quad (\text{Lemma 2.1.14})$$

i) folgt aus 1'), iii) aus 4'). ii) ergibt sich aus 2') und 3'), da nach Lemma 2.1.20 folgendes gilt:

$$\forall i, j \in \mathbb{N}_m (i \neq j \Rightarrow \underline{x}_i \wedge \underline{x}_j = \underline{0}) \wedge \left(\bigvee_{i \in \mathbb{N}_m} \underline{x}_i = \underline{1} \right) \Leftrightarrow \sum_{i \in \mathbb{N}_m} \underline{x}_i = \underline{1}$$

$$\Leftrightarrow \forall j \in \mathbb{N}_n \left(\sum_{i \in \mathbb{N}_m} x_{ij} = 1 \right). \quad \square$$

Definition 2.1.22

Eine quadratische Programmierungsaufgabe (kurz, Problem QP) bezüglich des Problems $P = (NG, a, Z)$ wird folgendermaßen angegeben:

$$\text{Maximiere } Z = \frac{1}{2} \sum_{i \in \mathbb{N}_m} \sum_k \sum_{\substack{l \\ k, l \in \mathbb{N}_n}} f_{kl} x_{ik} x_{il}$$

unter den folgenden Bedingungen:

$$\forall i, j \in \mathbb{N}_n \quad (x_{ij} \in \{0, 1\}) \quad \text{R1}$$

$$i \in \mathbb{N}_m \quad \left(\sum_{j \in \mathbb{N}_n} x_{ij} \neq 0 \right) \quad \text{R2}$$

$$j \in \mathbb{N}_n \quad \left(\sum_{i \in \mathbb{N}_m} x_{ij} = 1 \right) \quad \text{R3}$$

$$i \in \mathbb{N}_m \quad \left(\sum_{j \in \mathbb{N}_n} g_j x_{ij} \leq a \right) \quad \text{R4}$$

R1, R2, R3 und R4 heißen Restriktionen von QP. Sei $U = \{x_i \mid i \in \mathbb{N}_m\}$.

$(U, W(U))$ ist g.d. eine Lösung von QP, wenn $\forall x_i \in U$ den Restriktionen R1-R4 genügen und $W(U) = \hat{Z}$. $L' = \{(U, W(U)) \mid (U, W(U)) \text{ ist eine Lösung von QP}\}$ heißt Lösungsmenge von QP.

Satz 2.1.23

$(U, W(U))$ ist eine Lösung von QP $\iff (B, Z(B))$ ist eine Lösung von P.

Beweis:

Nach Satz 2.1.21 ist B genau dann eine zulässige Zerlegung, wenn alle $x_i \in U$ den Restriktionen R1, R2, R3 und R4 genügen. Nach Lemma 2.1.19 gilt $Z = W(U) = Z(B)$. $Z(B)$ ist genau dann das Optimum von P, wenn $W(U) = \hat{Z}$. $\square$

Korollar 2.1.24

Das Problem P ist genau dann gelöst, wenn die Lösungen von QP bekannt sind.

Beweis:

Die Aussage ist eine unmittelbare Folgerung des Satzes 2.1.23. $\square$

Satz 2.1.25

Sei $NZ_B = (B, K, q, h)$ ein durch B induziertes Netz. Seien $b_i, b_j \in B$.

NZ_B ist eine optimale Zerlegung von $NG \Leftrightarrow \sum_{\{b_i, b_j\} \in K} h(\{b_i, b_j\})$ ist minimal.

Beweis:

Es ist zu beweisen:

$$\text{Max } Z(B) = \text{Min } \frac{1}{2} \sum_{\{b_i, b_j\} \in K} h(\{b_i, b_j\}).$$

$$\text{Sei } F = \frac{1}{2} \sum_{\substack{k=1 \\ k, l \in \mathbb{N}_n}} f_{kl}.$$

$$\sum_{\{b_i, b_j\} \in K} h(\{b_i, b_j\}) = \sum_{\{b_i, b_j\} \in K} \sum_{\substack{k \in b_i \\ l \in b_j \\ \{k, l\} \in E}} f_{kl} \quad (\text{Def. 2.1.11})$$

$$= \sum_{\{b_i, b_j\} \in K} \sum_k \sum_{\substack{l=1 \\ k, l \in \mathbb{N}_n}} f_{kl} x_{ik} x_{jl} \quad (\text{Lemma 2.1.15})$$

$$= \sum_{i \in \mathbb{N}_m} \sum_k \sum_{\substack{l=1 \\ k, l \in \mathbb{N}_n}} f_{kl} x_{ik} (1 - x_{il})$$

$$= \sum_k \sum_{\substack{l=1 \\ k, l \in \mathbb{N}_n}} f_{kl} \left(\sum_{i \in \mathbb{N}_m} x_{ik} - \sum_{i \in \mathbb{N}_m} x_{ik} x_{il} \right)$$

$$= \sum_k \sum_{\substack{l=1 \\ k, l \in \mathbb{N}_n}} f_{kl} \left(1 - \sum_{i \in \mathbb{N}_m} x_{ik} x_{il} \right) \quad \left(\sum_{i \in \mathbb{N}_m} x_{il} = 1 \right)$$

$$\frac{1}{2} \sum_{\{b_i, b_j\} \in K} h(b_i, b_j) = F - Z(B).$$

Da F für ein gegebenes Netz eine Konstante ist, so folgt diese Aussage. $\square$

Bemerkung 2.1.26

Die Zielfunktion Z zu maximieren ist also identisch mit der Minimierung der Übergangshäufigkeiten zwischen den einzelnen Blöcken. In den darauffolgenden Abschnitten werden wir uns weiterhin mit der Maximierung der Zielfunktion Z beschäftigen, da sie mathematisch einfacher zu handhaben ist.

Bei den vorangehenden Diskussionen wurde angenommen, daß ein $m \in \mathbb{N}$ existiert, das die Anzahl der Blöcke einer optimalen Zerlegung angibt. Für das Problem QP ist m am Anfang eine unbekannte Größe, die sich erst ergibt, nachdem die Lösung des Problems vorliegt. Es stellt sich nun die Frage, wie m abzuschätzen ist.

Satz 2.1.27

Für das Problem QP gilt die Relation:

$$\left\lceil \sum_{j \in \mathbb{N}_n} g_j / a \right\rceil \leq m \leq n, \quad ,$$

wobei $\lceil x \rceil$ = die kleinste natürliche Zahl, die $\geq x$ ist.

Beweis:

$$\text{i) Fall } \left\lceil \sum_{j \in \mathbb{N}_n} g_j / a \right\rceil \leq m.$$

$$\sum_{j \in \mathbb{N}_n} g_j x_{ij} \leq a \quad (\text{Def. 2.1.22 R4})$$

$$\sum_{i \in \mathbb{N}_m} \sum_{j \in \mathbb{N}_n} g_j x_{ij} \leq ma$$

$$\sum_{j \in \mathbb{N}_n} g_j \sum_{i \in \mathbb{N}_m} x_{ij} \leq ma$$

$$\sum_{j \in \mathbb{N}_n} g_j \leq ma \quad (\text{Def. 2.1.22 R3})$$

$$\left\lceil \sum_{j \in \mathbb{N}_n} g_j / a \right\rceil \leq m$$

ii) Fall $m \leq n$.

$$\forall i \in \mathbb{N}_m : \left(\sum_{j \in \mathbb{N}_n} g_j x_{ij} \leq a \wedge \sum_{j \in \mathbb{N}_n} x_{ij} \neq 0 \right) \Rightarrow a \geq \max \{g_1, \dots, g_n\}.$$

Dies bedeutet, daß in jedem Block mindestens ein Knoten enthalten sein muß. Daraus folgt $m \leq n$. $\square$

Bezeichnung 2.1.28

Mit $m_u = \left\lceil \sum_{j \in \mathbb{N}_n} g_j / a \right\rceil$ wird die untere Schranke, $m_o = n$ die obere Schranke von m bezeichnet.

Definition 2.1.29

Sei $\hat{B}$ eine optimale Zerlegung mit $|\hat{B}| = m$.

$\alpha \hat{=} m/m_u$ heißt Ausdehnungsgrad von $\hat{B}$.

$\beta \hat{=} a/\bar{g}$ heißt relative Blockkapazität der Blöcke in $\hat{B}$, wobei

$$\bar{g} = \frac{1}{n} \sum_{j \in \mathbb{N}_n} g_j \text{ das mittlere Knotengewicht ist.}$$

Satz 2.1.30

$$i) 1 \leq \alpha \leq m_o/m_u$$

$$ii) 1 \leq \alpha \leq \beta$$

Beweis:

i) Die Aussage ergibt sich aus Satz 2.1.27.

$$ii) 1 \leq \alpha \leq n/m_u = n / \left\lceil \sum_{j \in \mathbb{N}_n} g_j / a \right\rceil$$

$$\leq n / \sum_{j \in \mathbb{N}_n} g_j / a$$

$$= a / \left(\frac{1}{n} \sum_{j \in \mathbb{N}_n} g_j \right)$$

$$= a/\bar{g} = \beta. \quad \square$$

$$\left(\sum_{j \in \mathbb{N}_n} g_j / a \leq \left\lceil \sum_{j \in \mathbb{N}_n} g_j / a \right\rceil \right)$$

Bemerkung 2.1.31

Der relativen Blockkapazität β entsprechend läßt sich a , die Gewichtsgrenze, als absolute Blockkapazität betrachten. β spielt eine wichtige Rolle für die Bestimmung des Rechenaufwands des noch anzugebenden Rechenverfahrens zur Lösung des Problems P. Der Ausdehnungsgrad α gibt an, wie gut die verfügbare relative Blockkapazität bei einer optimalen Zerlegung ausgenutzt wird (je kleiner α ist, desto besser ist die Ausnutzung).

Theoretisch läßt sich das Problem QP mittels des von P.L. Hammer und S. Rudeanu entwickelten sogenannten pseudoboolischen Programmierungsverfahrens /58/ lösen. Ihre Lösungsmethode besteht darin, das Prinzip der dynamischen Programmierung mit der von ihnen entwickelten Methode zur Auffindung des lokalen Optimums einer pseudoboolischen Funktion zu kombinieren. Zur Lösung des Problems QP muß zunächst angenommen werden, daß $m = n$. Auf diese Weise enthält das Problem n^2 0 - 1 Variable, eine quadratische Zielfunktion und $3n$ lineare pseudoboolische Funktionen. Das Verfahren ist, wie bereits in Abschnitt 1.2 erwähnt, wegen seines großen Rechenaufwands für die meisten in der Praxis auftretenden Probleme nicht anwendbar. Im nächsten Abschnitt werden wir ein Konzept zur Lösung des Problems angeben, auf dessen Grundlage ein mathematisch fundiertes, praktisch brauchbares neues Rechenverfahren entwickelt wird.

2.2 Das Angsche Konzept zur Lösung des Problems

Das Konzept beruht auf einer scheinbar sehr einfachen Idee: "divide et impera" ("teile und herrsche"). Diese Idee wird seit der Entstehung der Menschheit immer wieder von den Menschen benutzt, um große und schwierige Probleme zu lösen. Zu dieser Idee gibt es ein uraltes chinesisches Gleichnis, die Parabel "Yü Gung versetzt Berge". Darin wird erzählt, daß in alten Zeiten im Norden Chinas ein Greis aus den Nördlichen Bergen namens Yü Gung ("Närrischer Greis") lebte. Den Weg, der von seiner Haustür nach Süden führte, versperrten zwei große Berge. Yü Gung faßte den Entschluß, gemeinsam mit seinen Söhnen diese Berge mit Hacken und Schaufeln abzutragen. Ein anderer Greis namens Dschi Sou ("Weiser Alter") lachte, als er sie bei der Arbeit sah, und meinte: "Ihr treibt aber wirklich Unfug; ihr paar Leute könnt doch unmöglich zwei solch riesige Berge abtragen!". Yü Gung antwortete ihm: "Sterbe ich, bleiben meine Kinder, sterben die Kinder, bleiben die Enkelkinder, und so werden sich die Generationen in einer endlosen Reihe ablösen. Diese Berge sind zwar hoch, sie können jedoch nicht mehr höher werden. Um das, was wir abtragen, werden sie niedriger. Warum sollten wir sie da nicht abtragen können?" Nachdem Yü Gung mit diesen Worten die falsche Auffassung Dschi Sous widerlegt hatte, machte er sich daran - ohne auch nur im geringsten zu schwanken -, Tag für Tag die Berge abzutragen. Das rührte Gott, und er schickte zwei seiner Boten auf die Erde, die beide Berge auf dem Rücken davontrugen.

"Berge abzutragen" war also das große, schwierige Problem Yü Gungs. Seine Idee zur Lösung des Problems bestand darin, die Berge Stück für Stück mit Hacken und Schaufeln abzutragen. Dabei betrachtete er "ein Stück Berg abtragen" als ein Teilproblem, das von ihm und von jedem einzelnen seiner Nachkommenschaft mit allen damals verfügbaren Werkzeugen bewältigt werden sollte. Er war wahrscheinlich von der - typisch asiatischen - Überzeugung ausgegangen, daß sich seine Nachkommenschaft Generation für Generation so weiter vermehre, daß eines Tages alle Teilprobleme doch gelöst werden könnten. Im übertragenen Sinne läßt sich die Idee Yü Gungs als eine Enumerationsmethode /275,281,282/ zur Lösung allgemeiner kombinatorischer Optimierungsaufgaben betrachten, wobei eine gewisse Analogie zwischen dem Stammbaum der Familie Yü Gungs und dem Enumerationsbaum besteht: Jeder Knoten des Baumes kann als ein Teilproblem aufgefaßt werden. Von jedem Knoten aus wird das entsprechende Teilproblem weiter aufgeteilt; die sich ergebenden

Teilprobleme bilden dann die Nachfolgerknoten. Ohne die Gottes Boten, die als Ideen bzw. Strategien zur Aufteilung des Problems bzw. zur Beschleunigung des Enumerationsprozesses aufgefaßt werden, wäre die Enumerationsmethode eine Vollenumeration ("Brute Force Methode" /282/), mit deren Hilfe das Problem kaum in absehbarer Zeit gelöst werden könnte.

In diesem Abschnitt werden zunächst zwei Möglichkeiten zur Aufteilung des Problems P gegeneinander abgewogen. Die dadurch entstehenden Teilprobleme werden definiert. In den darauffolgenden Abschnitten werden Rechenverfahren, die als Prozesse formuliert werden, zur Lösung der Teilprobleme entwickelt und begründet. Anschließend werden praktische Beispiele angegeben.

Zur Aufteilung des Problems $P = (NG, a, Z)$ mit $NG = (V, E, g, f)$ bieten sich die folgenden zwei brauchbaren Vorschläge an:

Vorschlag A

Teilproblem A1

Gegeben: Die Menge V mit $|V| = n$.

Gesucht: $T =$ Die Menge aller Zerlegungen von V .

Teilproblem A2

Gegeben: T und (g, a) .

Gesucht: $\pi \subseteq T$, wobei $\pi =$ die Menge aller zulässigen Zerlegungen von V .

Teilproblem A3

Gegeben: π , f und Z .

Gesucht: $L =$ Die Lösungsmenge des Problems P .

Vorschlag B

Teilproblem B1

Gegeben: V und (g, a) .

Gesucht: $S =$ Die Menge aller nicht leeren Teilmengen von V , die der Bedingung $\sum_{j \in \mathbb{N}_n} g_j \leq a$ genügen.

Teilproblem B2

Gegeben: S

Gesucht: $\pi \in 2^S \setminus \{\emptyset\}$, wobei π = die Menge aller zulässigen Zerlegungen von V.Teilproblem B3Gegeben: π , f und Z.

Gesucht: L = Die Lösungsmenge des Problems P.

Die Frage lautet nun: Welcher Vorschlag ist besser?

Aus der Definition der Teilprobleme ist es leicht einzusehen, daß A3 identisch ist mit B3. Bei den Teilproblemen A2 und B2 handelt es sich darum, bestimmte Teilmengen einer gegebenen Menge aufzufinden. Beide Probleme können mittels der Enumerationsmethode gelöst werden. Der Rechenaufwand wird von der Mächtigkeit der Menge T bzw. der Menge S bestimmt, wobei sich T aus A1, S aus B1 ergibt.

Satz 2.2.1 (Satz von Dobinski)

Sei M eine endliche Menge mit $|M| = n$. Sei $p(n)$ die Anzahl der Zerlegungen von M.

$$i) \quad p(n+1) = \sum_{i=0}^n \binom{n}{i} p(i)$$

$$ii) \quad p(n) = \frac{1}{e} \sum_{i=0}^{\infty} \frac{i^n}{i!}.$$

Beweis:

Der Beweis dieses Satzes ist z.B. in /281/ angegeben. $\square$ Korollar 2.2.2

$$i) \quad p(0) = p(1) = 1$$

$$ii) \quad p(n+1) > p(n) > 1 \quad \text{für } n \geq 2.$$

Beweis: Die Aussagen sind unmittelbare Folgerungen des Satzes 2.2.1. $\square$

Bemerkung 2.2.3

$|T| = p(n)$ und $|S| \leq 2^n - 1$. Damit ist ein Vergleich zwischen $|T|$ und $|S|$ möglich.

Satz 2.2.4

$$p(n) > 2^n \quad \text{für } \forall n \geq 5.$$

Beweis:

Man überzeugt sich leicht davon, daß

$$p(n) < 2^n \quad \text{für } 0 \leq n \leq 4.$$

Vollständige Induktion über n für $n \geq 5$.

$$p(5) = 52 > 2^5$$

Sei $p(n) > 2^n$.

$$p(n+1) = \sum_{i=0}^n \binom{n}{i} p(i) \quad (\text{Satz 2.2.1})$$

$$= \sum_{i=0}^{n-1} \binom{n}{i} p(i) + p(n)$$

$$> \sum_{i=0}^{n-1} \binom{n}{i} p(i) + 2^n$$

$$> 2^n + 2^n$$

$$= 2^{n+1}. \quad \square$$

$$\left(\sum_{i=0}^{n-1} \binom{n}{i} \right) = 2^n - 1,$$

Korollar 2.2.2)

Es gilt also nach Satz 2.2.4 $|T| > |S|$ für $n \geq 5$. In Wirklichkeit wächst $p(n)$ mit zunehmendem n sehr viel schneller als 2^n , wie die Tabelle für $p(n)$ und 2^n in Anhang C zeigt. Für $n = 15$ ist $p(n)$ bereits so groß, daß die Teilprobleme A1 und A2 auch mit Hilfe einer modernen Rechenanlage nicht mit vertretbarem Aufwand gelöst werden können. Der Vorschlag A bleibt aus diesem Grunde für die weiteren Untersuchungen außer Betracht.

In Teilproblem B1 wird S von den Parametern V , g und a bestimmt. $|S|$ nimmt i.a. mit wachsendem n exponentiell zu, sodaß für großes n eine obere Grenze erreicht wird, die diesen Vorschlag schließlich unattraktiv erscheinen läßt,

falls keine effektiven Maßnahmen getroffen werden. Man fragt sich nun, ob alle Elemente von S für die Bestimmung des Optimums unentbehrlich sind und, falls diese Frage verneint wird, wie die zur Lösung der Teilprobleme B2 und B3 belanglosen Elemente von S aufgefunden werden können. Zur Beantwortung dieser Frage betrachten wir zunächst das Problem QP. Es wird untersucht, wie QP gemäß dem Vorschlag B gelöst werden kann. Um eine bessere mathematische Handhabung zu ermöglichen, werden wir zunächst ein äquivalentes Problem von QP definieren, in dem ausschließlich die charakteristischen Vektoren als Gegenstand der Betrachtung enthalten sind.

Definition 2.2.5

Sei B eine zulässige Zerlegung von V . Sei $\underline{x} = (x_1, \dots, x_n) \in \{0, 1\}^n$.

Sei $I_{\underline{x}} = \{j \mid x_j = 1\}$.

$U \equiv \{\underline{x} \mid \underline{x} = \text{charak. Vektor von } b \in B\}$ heißt charakteristische Menge von B .

$w(\underline{x}) \equiv \frac{1}{2} \sum_k \sum_{l \in I_{\underline{x}}} f_{kl} x_k x_l = \frac{1}{2} \sum_k \sum_{l \in \mathbb{N}_n} f_{kl} x_k x_l$ heißt Wert von $\underline{x}$.

Sei $\underline{x} \in U$. Mit $g_{\underline{x}}$ wird die Summe $\sum_{j \in \mathbb{N}_n} g_j x_j$ bezeichnet.

Definition 2.2.6

Sei B eine zulässige Zerlegung von V . Sei U die charakteristische Menge von B .

Das Problem Q wird folgendermaßen angegeben:

$$\text{Maximiere } W(U) = \sum_{\underline{x} \in U} w(\underline{x})$$

mit den Restriktionen:

$$\underline{x} \in \{0, 1\}^n \setminus \{0\} \quad R_1$$

$$\bigvee_{\underline{x} \in U} \underline{x} = \underline{1} \quad R_2$$

$$\forall \underline{x}, \underline{y} \in U (\underline{x} \neq \underline{y} \implies \underline{x} \wedge \underline{y} = \underline{0}) \quad R_3$$

$$g_{\underline{x}} \leq a \quad R_4$$

$(U, W(U))$ ist g.d. eine Lösung von Q , wenn $\forall \underline{x} \in U$ den Restriktionen $R_1 - R_4$ genügen und $W(U) = \hat{Z}$. $L'' \equiv \{(U, W(U)) \mid (U, W(U)) = \text{Lösung von } Q\}$ heißt Lösungs-Menge von Q .

Satz 2.2.7

$$Q \approx QP$$

Beweis:

Q ist eine neue Formulierung von QP, die gemäß den neuen Bezeichnungen in Definition 2.2.5 festgelegt wird. Man überzeugt sich leicht von der Gültigkeit der folgenden Aussagen:

$$W(U) = Z$$

$$R_1 \iff R1 \wedge R2$$

$$R_2 \wedge R_3 \iff R3 \quad (\text{Lemma 2.1.20})$$

$$R_4 \iff R4$$

Q und QP besitzen also die gleiche Lösungsmenge. $\square$

Wir werden künftig alle bekannten bzw. gegebenen Größen als Parameter bezeichnen. Das Problem Q läßt sich gemäß dem Vorschlag B folgendermaßen aufteilen:

Teilproblem Q1

Parameter: V, g, a und f.

Gesucht: Die Mengen C und D, wobei

$$C \equiv \{ \underline{x} \mid \underline{x} \in \{0,1\}^n \setminus \{0\} \wedge g\underline{x} \leq a \}$$

$$D \equiv \{ w(\underline{x}) \mid \underline{x} \in C \}.$$

Teilproblem Q2

Parameter: C und D.

Gesucht: Die Menge H und F, wobei

$$H \equiv \{ U \mid U \in 2^C \wedge (\bigvee_{\underline{x} \in U} \underline{x} = \underline{1}) \wedge \forall \underline{x}, \underline{y} \in U (\underline{x} \neq \underline{y} \implies \underline{x} \wedge \underline{y} = \underline{0}) \}$$

$$F \equiv \{ W(U) \mid U \in H \}.$$

Teilproblem Q3

Parameter: H und F.

Gesucht: $\hat{Z}$ und $\hat{U}$, wobei

$$\hat{Z} \triangleq \text{Max } F$$

$$\hat{U} \triangleq \{U \mid U \in H \wedge W(U) = \hat{Z}\}.$$

Satz 2.2.8

$L = \hat{U} \times \{\hat{Z}\}$ ist die Lösungsmenge von Q.

Beweis:

Diese Aussage ergibt sich aus der Konstruktion der Teilprobleme Q1, Q2 und Q3. $\square$

Nach Definition 2.2.5 gilt $w(\underline{x}) \geq 0$ für $\forall \underline{x} \in C$. Um $W(U)$ zu maximieren, benötigt man ausschließlich diejenigen $\underline{x}$, bei denen $w(\underline{x}) > 0$. Dies bedeutet, daß sich die Menge C folgendermaßen zerlegen läßt:

$$C_1 \triangleq \{\underline{x} \mid \underline{x} \in C \wedge w(\underline{x}) > 0\}.$$

$$C_2 \triangleq C \setminus C_1.$$

Lemma 2.2.9

$$C \neq \emptyset \iff C_2 \neq \emptyset.$$

Beweis:

" $\implies$ " $C \neq \emptyset$ bedeutet, daß es mindestens ein $\underline{x}$ mit $|I_{\underline{x}}| = 1$ gibt.

Da $|I_{\underline{x}}| = 1 \implies w(\underline{x}) = 0$ (Def. 2.2.5), so folgt diese Aussage.

" $\impliedby$ " Offensichtlich. $\square$

Für $C \neq \emptyset$ gibt es zwei Möglichkeiten:

i) $C_1 = \emptyset$

ii) $C_1 \neq \emptyset$.

Lemma 2.2.10

$$C_1 = \emptyset \iff \hat{Z} = 0$$

Beweis: Offensichtlich. $\square$

Bemerkung 2.2.11

Wenn $C_1 = \emptyset$, dann ergibt sich $\hat{U}$ aus Q2. Dieser Fall tritt jedoch sehr selten ein.

Lemma 2.2.12

$$C_1 \neq \emptyset \iff \hat{Z} > 0.$$

Beweis:

$$\begin{aligned} C_1 \neq \emptyset &\iff \exists \underline{x} (g\underline{x} \leq a \wedge w(\underline{x}) > 0) \\ &\iff \exists U (W(U) > 0) \\ &\iff \hat{Z} > 0. \quad \square \end{aligned}$$

Lemma 2.2.13

Sei $C_1 \neq \emptyset$. Sei $(U, \hat{Z})$ eine Lösung des Problems Q. Sei $U = U' \cup U''$, wobei für U' und U'' folgendes gilt:

$$\begin{aligned} U' &= \{ \underline{x} \mid \underline{x} \in U \wedge \underline{x} \in C_1 \} \\ U'' &= \{ \underline{x} \mid \underline{x} \in U \wedge \underline{x} \in C_2 \}. \end{aligned}$$

$$\text{i) } \hat{Z} = \sum_{\underline{x} \in U'} w(\underline{x})$$

$$\text{ii) } \bigvee_{\underline{x} \in U'} \underline{x} = 1 \oplus \bigvee_{\underline{x} \in U''} \underline{x}$$

$$\text{iii) } U'' \neq \emptyset \iff \bigvee_{\underline{x} \in U'} \underline{x} \neq 1.$$

Beweis:

$$\text{i) } \hat{Z} = W(U) = \sum_{\underline{x} \in U'} w(\underline{x}) + \sum_{\underline{x} \in U''} w(\underline{x}) = \sum_{\underline{x} \in U'} w(\underline{x}). \quad \left(\sum_{\underline{x} \in U''} w(\underline{x}) = 0 \right)$$

$$\begin{aligned}
 \text{ii) } \underline{1} &= \bigvee_{\underline{x} \in U''} \underline{x} = \bigvee_{\underline{x} \in U'} \underline{x} \vee \bigvee_{\underline{x} \in U''} \underline{x} & (R_2) \\
 &= \bigvee_{\underline{x} \in U'} \underline{x} \oplus \bigvee_{\underline{x} \in U''} \underline{x} \oplus \left(\bigvee_{\underline{x} \in U'} \underline{x} \wedge \bigvee_{\underline{x} \in U''} \underline{x} \right) \quad (\underline{x} \vee \underline{y} = \underline{x} \oplus \underline{y} \oplus (\underline{x} \wedge \underline{y})) \\
 &= \bigvee_{\underline{x} \in U'} \underline{x} \oplus \bigvee_{\underline{x} \in U''} \underline{x}. & (R_3)
 \end{aligned}$$

$$\begin{aligned}
 \underline{1} \oplus \bigvee_{\underline{x} \in U''} \underline{x} &= \bigvee_{\underline{x} \in U'} \underline{x} \oplus \bigvee_{\underline{x} \in U''} \underline{x} \oplus \bigvee_{\underline{x} \in U''} \underline{x} \\
 &= \bigvee_{\underline{x} \in U'} \underline{x} & (\underline{x} \oplus \underline{x} = \underline{0})
 \end{aligned}$$

$$\begin{aligned}
 \text{iii) } U'' \neq \emptyset &\iff \bigvee_{\underline{x} \in U''} \underline{x} \neq \underline{0} \\
 &\iff \bigvee_{\underline{x} \in U'} \underline{x} \neq \underline{1} \cdot \square & \left(\bigvee_{\underline{x} \in U'} \underline{x} = \underline{1} \oplus \bigvee_{\underline{x} \in U''} \underline{x} \right)
 \end{aligned}$$

Nach Lemma 2.2.13 wird $\hat{Z}$ ausschließlich von U' bestimmt. Dies bedeutet, daß die Betrachtung der Menge C_1 allein hinreichend ist, um $\hat{Z}$ aufzufinden. Dabei wird C_2 vorläufig außer Acht gelassen. Ist $U'' \neq \emptyset$, dann gilt

$$\bigvee_{\underline{x} \in U'} \underline{x} \neq \underline{1},$$

d.h. die Restriktion R_2 wird nicht erfüllt. In diesem Falle wird die Menge C_2 wieder in Betracht gezogen. Mit C_2 kann nun U'' aufgefunden werden. Dabei genügt $U = U' \cup U''$ der Restriktion R_2 .

Da $|C_1| \ll |C|$ für viele in der Praxis auftretenden Probleme gilt, wird der Rechenaufwand zur Lösung der darauffolgenden Probleme stark reduziert. Der Preis, den man dafür zahlen muß, ist der zusätzliche Rechenaufwand zur Auffindung der Menge U'' . Die Menge U'' aufzufinden ist ein neues Teilproblem, bei dem jedoch kein exaktes, aufwendiges Lösungsverfahren benötigt wird, da sein Ergebnis keinen Einfluß auf das Optimum des Problems Q hat. Diese Überlegung führt zu einer neuen Aufteilung des Problems Q , die folgendermaßen angegeben wird:

Teilproblem Q₁

Parameter: V, g, a und f .

Gesucht: Die Mengen C_1, C_2 und D_1 , wobei

$$C_1 \hat{=} \{ \underline{x} \mid \underline{x} \in \{0,1\}^n \setminus \{0\} \wedge g\underline{x} \leq a \wedge w(\underline{x}) > 0 \}$$

$$C_2 \hat{=} \{ \underline{x} \mid \underline{x} \in \{0,1\}^n \setminus \{0\} \wedge g\underline{x} \leq a \wedge w(\underline{x}) = 0 \}$$

$$D_1 \hat{=} \{ w(\underline{x}) \mid \underline{x} \in C_1 \}.$$

Teilproblem Q₂

Parameter: $C_1 \neq \emptyset, D_1 \neq \emptyset$

Gesucht: $\hat{Z}$ und $\hat{U}'$, für die folgendes gilt:

$$\hat{Z} = \text{Max} \{ w(U') \mid U' \in H' \}$$

$$\hat{U}' = \{ U' \mid U' \in H' \wedge w(U') = \hat{Z} \}, \text{ wobei}$$

$$H' = \{ U' \mid U' \in 2^{C_1} \wedge \forall \underline{x}, \underline{y} \in U' (\underline{x} \neq \underline{y} \Rightarrow \underline{x} \wedge \underline{y} = 0) \}.$$

Teilproblem Q₃

Parameter: $C_2 \neq \emptyset, \hat{U}'$

Gesucht: Die Menge $\hat{U}''$, für die gilt:

$$\hat{U}'' = \{ U'' \mid U'' \in H'' \wedge \exists U' (U' \in \hat{U}' \wedge \bigvee_{\underline{x} \in U''} \underline{x} = \underline{1} \oplus \bigvee_{\underline{x} \in U'} \underline{x}) \},$$

$$\text{wobei } H'' = \{ U'' \mid U'' \in 2^{C_2} \wedge \forall \underline{x}, \underline{y} \in U'' (\underline{x} \neq \underline{y} \Rightarrow \underline{x} \wedge \underline{y} = 0) \}.$$

Auf der Grundlage dieser neuen Problemaufteilung läßt sich das Konzept zur Lösung des Problems Q folgendermaßen angeben:

Schritt 1: Problem Q₁ wird gelöst.

Wenn $C_1 = \emptyset$, dann $\hat{Z} = 0$ und $\hat{U}' = \emptyset$. Sprung nach 3.

Schritt 2: Problem Q₂ wird gelöst.

Wenn für $\forall U' \in \hat{U}' \bigvee_{\underline{x} \in U'} \underline{x} = \underline{1}$ gilt, dann $\hat{U} = \hat{U}'$. Ende.

Schritt 3: Problem Q₃ wird gelöst. Ende.

Ergebnis: $L'' = \hat{U} \times \{\hat{Z}\}$ ist die Lösungsmenge des Problems Q, wobei $\hat{U} = \hat{U}' \cup \hat{U}''$.

Beweis:

Fall 1 $C_1 = \emptyset$.

Es folgt aus Lemma 2.2.10 und Def. des Teilproblems Q_2 , daß $\hat{Z} = 0$ und $\hat{U}' = \{\emptyset\}$. Dies bedeutet: $\hat{U} = \hat{U}''$.

$\forall \hat{U}'' \in \hat{U}''$ genügt den Restriktionen des Problems Q. In diesem Falle ist $\hat{U} \times \{0\}$ die Lösungsmenge von Q.

Fall 2 $C_1 \neq \emptyset$.

Es folgt aus Lemma 2.2.12, daß $\hat{Z} > 0$, wobei $\hat{Z}$ (Lemma 2.2.13) bei Schritt 2 aufgefunden wird. Da $\hat{U}' \subseteq H'$, so genügt $\forall U' \in \hat{U}'$ den Restriktionen R_1, R_3 und R_4 .

Falls $\forall U' \in \hat{U}' (\sum_{x \in U'} x = 1)$, dann wird Schritt 3 nicht ausgeführt. Es gilt $\hat{U} = \hat{U}'$.

Falls $\exists U' \in \hat{U}'$ sodaß $\sum_{x \in U'} x \neq 1$, dann gibt es nach Lemma 2.2.13 ein $U'' \in \hat{U}''$,

sodaß für $U = U' \cup U''$ $\sum_{x \in U} x = 1$ gilt. In den letzten beiden Fällen genügt jedes $U \in \hat{U}$ der Restriktion R_2 . $\square$

Bemerkung 2.2.14

Die Menge $\hat{U}' \times \{\hat{Z}\}$ kann wegen $\hat{U}' \subseteq \hat{U}$ als die Lösungsmenge eines schwach äquivalenten Problems von Q betrachtet werden. Schließen wir den trivialen Fall $C_1 = \emptyset$ aus, dann kann man sich das Konzept folgendermaßen vorstellen: Statt Q wird zunächst ein schwach äquivalentes Problem Q' von Q gelöst, das aus den Teilproblemen Q_1 und Q_2 besteht. Ist $\hat{U}' \times \{\hat{Z}\}$ die Lösungsmenge von Q' und es gilt $\forall U' \in \hat{U}' (\sum_{x \in U'} x = 1)$, dann ist $\hat{U}' \times \{\hat{Z}\}$ auch die Lösungsmenge von Q. Sonst muß Q_3 gelöst werden. In der Praxis benötigt man nicht alle Lösungen von Q, sondern nur diejenigen Lösungen, die einen minimalen Ausdehnungsgrad besitzen. In diesem Falle ist es vorteilhafter, ausschließlich diejenigen $U'' \in H''$ aufzufinden, bei denen sich eine Lösung des Problems Q mit minimalem Ausdehnungsgrad ergibt. Dies ist eine lineare kombinatorische Optimierungsaufgabe, die zur Klasse der sogenannten "Bin Packing"-Probleme /79,80/ gehört, deren Komplexität als "NP-complete" klassifiziert wird. Zum Glück existieren bereits sehr gute approximierende Algorithmen für diese Klasse von Problemen.

Bemerkung 2.2.15

Wir werden künftig die Schritte 1, 2 und 3 als Rechenphasen (kurz, Phasen) des Konzepts bezeichnen.

Die Rechenverfahren zur Lösung der Teilprobleme Q_1 , Q_2 und Q_3 werden in den darauffolgenden Abschnitten entwickelt und begründet. In Abschnitt 2.3 wird zunächst eine Methode zur Lösung der linearen Ungleichung $gx \leq a$ angegeben, die auf rein kombinatorischen Überlegungen beruht. Dabei werden Begriffe wie Blockindikator, Indikatormenge, Indikatorfamilien usw. eingeführt, die in Abschnitt 2.5 graphentheoretisch interpretiert werden. Das Rechenverfahren zur Lösung des Problems Q_2 ist ein Enumerationsprozeß, der auf dem Verzweigungsprinzip /1 - 19/ beruht. Dabei werden Strategien zur Beschleunigung des Enumerationsprozesses entwickelt und gegeneinander abgewogen. Dieser Prozeß, der "binärer Verzweigungsprozeß" genannt wird, wird in Abschnitt 2.4 präzisiert. Es hat sich herausgestellt, daß in der Menge C_1 sehr viele Vektoren enthalten sind, die für die Bestimmung des Optimums in der Phase 2 keine Rolle spielen. Solche "unwesentlichen" Vektoren lassen sich von Anfang an ausschließen, wenn bei der Phase 1 die Strukturen der Blockteilgraphen der einzelnen Blöcke analysiert werden. Zu diesem Zweck wird der "Erweiterungsprozeß" entwickelt, der in Abschnitt 2.5 begründet wird. Mittels des Erweiterungsprozesses wird eine kleine Teilmenge von C_1 aufgefunden, in der ausschließlich "wesentliche" Vektoren enthalten sind. Es wird gezeigt, daß die Betrachtung dieser wesentlichen Vektoren allein hinreichend ist, um das Optimum in der Phase 2 aufzufinden. Dadurch wird der Rechenaufwand des binären Verzweigungsprozesses der Phase 2 stark reduziert. In Abschnitt 2.6 wird der "Ergänzungsprozeß" zur Lösung des Teilproblems Q_3 definiert. Dabei wird Q_3 als ein spezielles "Bin Packing"-Problem betrachtet. Für den Ergänzungsprozeß werden verschiedene Strategien entwickelt, die es ermöglichen, eine gute approximierende Lösung des derart formulierten Problems mit wenig Rechenaufwand aufzufinden. Der Erweiterungsprozeß, der binäre Verzweigungsprozeß und der Ergänzungsprozeß zusammen tragen dazu bei, das eben beschriebene Konzept zu einem Rechenverfahren zu entwickeln, das für die Lösung der in der Praxis auftretenden Probleme brauchbar ist. In Abschnitt 2.7 wird eine zusammenfassende Darstellung dieses Rechenverfahrens angegeben. In Abschnitt 2.8 werden schließlich einige an verschiedenen Literaturstellen angegebene Probleme mittels dieses Rechenverfahrens gelöst.

2.3 Der Blockindikator, die Indikatormenge und die Indikatorfamilien

Sei $NG = (V, E, g, f)$ ein Netz. Sei $g = (g_1, \dots, g_n)$. Sei B eine zulässige Zerlegung von V .

Definition 2.3.1

Die Relation

$VOR \subseteq V \times V$, angegeben durch

$$i \text{ VOR } j \iff (g_i > g_j) \vee (g_i = g_j \wedge i \leq j)$$

heißt Knotenordnungsrelation (kurz, Knotenrelation).

Folgerung 2.3.2

VOR ist eine lineare Ordnung auf V .

Definition 2.3.3

Sei $\underline{x} = (x_1, \dots, x_n)$ ein charakteristischer Vektor. Seien $i, j, k, l \in \mathbb{N}_n$.

$h = (h_1, \dots, h_n)$ ist genau dann eine durch VOR induzierte Knotengewichtung, wenn gilt:

$$\forall h_k \forall h_l (k \leq l \iff \exists i \exists j (h_k = g_i \wedge h_l = g_j \wedge i \text{ VOR } j)).$$

$\underline{s} = (s_1, \dots, s_n)$ heißt genau dann der zugeordnete Vektor von $\underline{x}$, wenn gilt:

$$\forall s_k \forall s_l (k \leq l \iff \exists i \exists j (s_k = x_i \wedge s_l = x_j \wedge i \text{ VOR } j)).$$

Mit $\underline{s}$ wird die Summe $\sum_{i \in \mathbb{N}_n} h_i s_i$ bezeichnet.

Bemerkung 2.3.4

h ergibt sich aus g , indem die Komponenten von g gemäß der Relation VOR linear geordnet werden. Damit erhält jede Komponente von g einen neuen Index, der ihren Platz in der linearen Ordnung kennzeichnet. Jede derart indizierte Komponente von g bildet eine Komponente von h mit demselben Index. Auf gleiche Weise wird $\underline{s}$ aus $\underline{x}$ gebildet.

Folgerung 2.3.5

- i) Für jedes g gibt es genau ein h .
- ii) Für jedes $\underline{x}$ gibt es genau ein $\underline{s}$.
- iii) $\forall i, j \in \mathbb{N}_n (i \leq j \iff h_i \geq h_j)$.

Lemma 2.3.6

Sei $\underline{s}$ der zugeordnete Vektor von $\underline{x}$.

- i) $I_{\underline{x}} = I_{\underline{s}}$
- ii) $w(\underline{x}) = w(\underline{s})$

Beweis:

Die Aussagen i) und ii) ergeben sich aus Definition 2.2.5 und Bemerkung 2.3.4. $\square$

Satz 2.3.7

Sei U die charakteristische Menge von B .

Sei $S \hat{=} \{ \underline{s} \mid \underline{s} \text{ ist der zugeordnete Vektor von } \underline{x} \in U \}$.

- i) $w(U) = \sum_{\underline{x} \in U} w(\underline{x}) = \sum_{\underline{s} \in S} w(\underline{s}) = w(S)$
- ii) $\bigvee_{\underline{x} \in U} \underline{x} = \underline{1} \iff \bigvee_{\underline{s} \in S} \underline{s} = \underline{1}$
- iii) $\forall \underline{x}, \underline{x}' \in U (\underline{x} \neq \underline{x}' \implies \underline{x} \wedge \underline{x}' = \underline{0}) \iff \forall \underline{s}, \underline{s}' \in S (\underline{s} \neq \underline{s}' \implies \underline{s} \wedge \underline{s}' = \underline{0})$
- iv) $g\underline{x} \leq a \iff h\underline{s} \leq a$

Beweis:

i) folgt aus Lemma 2.3.6. ii) und iii) ergeben sich aus Bemerkung 2.3.4 und Folgerung 2.3.5.

$$\text{iv) } \sum_{i \in \mathbb{N}_n} g_i x_i = \sum_{j \in \mathbb{N}_n} h_j s_j \quad . \quad \square$$

Definition 2.3.8

Das Problem QS wird folgendermaßen angegeben:

$$\text{Maximiere } W(S) = \sum_{\underline{s} \in S} w(\underline{s})$$

mit den Restriktionen:

$$\forall \underline{s} \in S (\underline{s} \in \{0,1\}^n \setminus \{0\}) \quad R_1^1$$

$$\bigvee_{\underline{s} \in S} \underline{s} = \underline{1} \quad R_2^1$$

$$\forall \underline{s}, \underline{s}' \in S (\underline{s} \neq \underline{s}' \implies \underline{s} \wedge \underline{s}' = \underline{0}) \quad R_3^1$$

$$hs \leq a \quad R_4^1$$

$(S, W(S))$ ist genau dann eine Lösung von QS, wenn $\forall \underline{s} \in S$ den Restriktionen $R_1^1 - R_4^1$ genügen und $W(S) = \hat{Z}$.

$LS = \{(S, W(S)) \mid (S, W(S)) \text{ ist eine Lösung von QS}\}$ heißt Lösungsmenge von QS.

Satz 2.3.9

$(U, W(U))$ ist eine Lösung von Q $\iff (S, W(S))$ ist eine Lösung von QS.

Beweis:

Diese Aussage ist eine unmittelbare Folgerung des Satzes 2.3.7. $\square$

Korollar 2.3.10

Das Problem Q ist genau dann gelöst, wenn die Lösungsmenge von QS aufgefunden ist.

Beweis:

Diese Aussage ergibt sich aus Satz 2.3.9. $\square$

Bemerkung 2.3.11

Aus einem bald ersichtlichen Grunde werden wir statt Q das Problem QS lösen. Für die Teilprobleme Q_1 , Q_2 und Q_3 bedeutet dies, daß statt x sein zugeordneter Vektor $\underline{s}$ betrachtet werden muß. Falls eine Lösung des Problems QS vorliegt, lassen sich die entsprechenden Elemente von U mittels der in Def. 2.3.3 angegebenen Vorschrift auffinden.

Definition 2.3.12

Sei $\underline{hs} = \sum_{i \in \mathbb{N}_n} h_i s_i \leq a$ eine lineare Ungleichung, für die gilt:

$$\sum_{i \in \mathbb{N}_n} h_i > a$$

und $a \geq h_1 - \dots - h_n > 0$.

$$G \equiv \left\{ (d_1, \dots, d_n) \mid (d_1, \dots, d_n) \in \{0, 1\}^n \wedge \sum_{i \in \mathbb{N}_n} h_i d_i \leq a \right\}$$

heißt Lösungsmenge von $\underline{hs} \leq a$. Jedes Element $\underline{d} \in G$ heißt eine Lösung.

Sei $j \in \mathbb{N}_n$.

$$G_j \equiv \left\{ \left(\underset{(j-1) \times}{(0, \dots, 0)}, 1, d_{j+1}, \dots, d_n \right) \mid (d_{j+1}, \dots, d_n) \in \{0, 1\}^{n-j} \wedge \sum_{i=j+1}^n h_i d_i \leq a - h_j \right\}$$

heißt die j-te Lösungsfamilie von $\underline{hs} \leq a$.

Folgerung 2.3.13

i) $\underline{hs} \leq a$ besitzt n Lösungsfamilien.

ii) $G_n = \{e_n\}$.

iii) $G_j \neq \emptyset$ für $\forall j \in \mathbb{N}_n$.

iv) $\forall i, j \in \mathbb{N}_n (i \neq j \Rightarrow G_i \cap G_j = \emptyset)$.

v) $G = \bigcup_{i \in \mathbb{N}_n} G_i \cup \{0\}$.

Bemerkung 2.3.14

Die hier definierte Lösungsfamilie hat keine Gemeinsamkeit mit der an manchen Literaturstellen (z.B. /58/) angegebenen sogenannten "Family of Solutions".

Bemerkung 2.3.15

Falls $\sum_{i \in \mathbb{N}_n} h_i \leq a$, dann ist $\underline{1}$ trivialerweise eine Lösung von $\underline{hs} \leq a$. Daraus

folgt, daß $w(\underline{1}) = \hat{Z}$. Wir werden diesen trivialen Fall von der weiteren Betrachtung ausschließen.

Für die Relation $\sum_{i=j+1}^n h_i d_i \leq a - h_j$ untersuchen wir die folgenden 5 Fälle:

Fall 1 : $h_j = a$.

Fall 2 : $h_j < a \wedge \sum_{i=j}^n h_i > a \wedge \exists k (k = \max \{ l \mid l > j \wedge h_j + h_l > a \})$.

Fall 3 : $h_j < a \wedge \sum_{i=j}^n h_i > a \wedge \exists k (k = \max \{ l \mid l > j \wedge h_j + h_l = a \})$.

Fall 4 : $h_j < a \wedge \sum_{i=j}^n h_i > a \wedge \forall k > j : (h_j + h_k < a)$.

Fall 5 : $h_j < a \wedge \sum_{i=j}^n h_i \leq a$.

Lemma 2.3.16

$$h_j = a \Rightarrow G_j = \{e_j\}$$

Beweis:

$$\begin{aligned} h_j = a &\Rightarrow \sum_{i=j+1}^n h_i d_i \leq 0 \\ &\Rightarrow d_i = 0 \text{ für } \forall i \in \{j+1, \dots, n\} \\ &\Rightarrow G_j = \{e_j\}. \quad \square \end{aligned}$$

Im folgenden sei $X_{k+1} \hat{=} \{(d_{k+1}, \dots, d_n) \mid (d_{k+1}, \dots, d_n) \in \{0, 1\}^{n-k} \wedge \sum_{i=k+1}^n h_i d_i \leq a - h_k\}$.

Lemma 2.3.17

$$\begin{aligned} h_j &< a \wedge \sum_{i=j}^n h_i > a \wedge \exists k (k = \max \{ l \mid l > j \wedge h_j + h_l > a \}) \\ &\Rightarrow G_j = \{(d_1, \dots, d_n) \mid \forall i \in \mathbb{N}_k \setminus \{j\} : (d_i = 0) \wedge d_j = 1 \wedge (d_{k+1}, \dots, d_n) \in X_{k+1}\}. \end{aligned}$$

Beweis:

Angenommen: Es gibt ein $\underline{d}' = (d'_1, \dots, d'_n) \in G_j$, sodaß

$$d'_1 = \dots = d'_{j-1} = 0 \wedge d'_j = 1 \wedge \exists l (j < l \leq k \wedge d'_l = 1 \wedge h_j + h_l > a).$$

Dies bedeutet, daß

$$\sum_{i=1}^n h_i d'_i = \sum_{i=1}^{j-1} h_i d'_i + h_j + \sum_{i=j+1}^{l-1} h_i d'_i + h_l + \sum_{i=l+1}^k h_i d'_i + \sum_{i=k+1}^n h_i d'_i$$

$$= h_j + h_1 + \sum_{i=k+1}^n h_i d_i' > a \quad \text{Widerspruch.} \quad \square$$

Lemma 2.3.18

$$h_j < a \wedge \sum_{i=j}^n h_i > a \wedge \exists k (k = \max \{ l \mid l > j \wedge h_j + h_l = a \})$$

$$\Rightarrow G_j = G_{j1} \cup G_{j2} \quad , \text{ wobei}$$

$$G_{j1} = \{ (d_1, \dots, d_n) \mid d_j = 1 \wedge d_k = 1 \wedge \forall i \in \mathbb{N}_n \setminus \{j, k\} : (d_i = 0) \}.$$

$$G_{j2} = \{ (d_1, \dots, d_n) \mid \forall i \in \mathbb{N}_n \setminus \{j\} : (d_i = 0) \wedge d_j = 1 \wedge (d_{k+1}, \dots, d_n) \in X_{k+1} \}.$$

Beweis:

G_{j1} enthält alle Lösungen, die zu G_j gehören und bei denen gilt: $d_j = d_1 = 1$.

Aus $\sum_{i \in \mathbb{N}_n \setminus \{j, 1\}} h_i d_i \leq a - h_j - h_1 = 0$ folgt $\forall i \in \mathbb{N}_n \setminus \{j, 1\} : (d_i = 0)$.

G_{j2} enthält alle Lösungen, die zu G_j gehören und bei denen gilt: $d_j = 1 \wedge d_1 = 0$.

Ist $(d_1, \dots, d_n)$ ein Element aus G_{j2} , so genügt $(d_{k+1}, \dots, d_n)$ der Ungleichung

$$\sum_{i=k+1}^n h_i d_i \leq a - h_j. \quad \text{D.h. } (d_{k+1}, \dots, d_n) \in X_{k+1}.$$

Nach Def. 2.3.12 wird G_j aus G_{j1} und G_{j2} gebildet. $\square$

Lemma 2.3.19

Seien $k, j \in \mathbb{N}_n$ und es gelte $k > j$.

Sei $G_{jk} \equiv \{ (d_1, \dots, d_n) \mid d_j = d_k = 1 \wedge \forall i \in \mathbb{N}_n \setminus \{j, k\} : (d_i = 0) \wedge (d_{k+1}, \dots, d_n) \in X_{k+1} \}$.

$$h_j < a \wedge \sum_{i=j}^n h_i > a \wedge \forall (k > j) : (h_j + h_k < a) \Rightarrow \bigcup_{k \in \mathbb{N}_n \setminus \mathbb{N}_j} G_{jk} \cup \{e_j\}.$$

Beweis:

Jede Menge G_{jk} mit $k \in \mathbb{N}_n \setminus \mathbb{N}_j$ ist eine Lösungsfamilie der linearen Ungleichung

$$\sum_{i=j+1}^n h_i d_i \leq a - h_j = c > 0. \quad (\text{LU})$$

Aus $h_j + h_k < a$ folgt $h_k < a - h_j = c$ für $\forall k \in \mathbb{N}_n \setminus \mathbb{N}_j$, sodaß die Bedingung

$$c > h_{j+1} \geq \dots \geq h_n > 0$$

erfüllt wird.

Es gibt insgesamt $(n - j)$ solche Lösungsfamilien von LU. Dazu muß noch das Element $\underline{e}_j$ für G_j berücksichtigt werden. Dies bedeutet, daß

$$G_j = \bigcup_{k \in \mathbb{N}_n \setminus \mathbb{N}_j} G_{jk} \cup \{\underline{e}_j\}. \quad \square$$

Lemma 2.3.20

- i) $h_j < a \wedge \sum_{i=j}^n h_i \leq a \Rightarrow G_j = \left\{ (d_1, \dots, d_n) \mid (d_1, \dots, d_j) = (0, \dots, 0, 1) \wedge (d_{j+1}, \dots, d_n) \in \{0, 1\}^{n-j} \right\}$.
- ii) $|G_j| = 2^{n-j}$.

Beweis:

- i) Jedes $(d_j, \dots, d_n) \in \{0, 1\}^{n-j}$ ist wegen $\sum_{i=j}^n h_i \leq a$ eine Lösung der linearen Ungleichung $\sum_{i=j}^n h_i d_i \leq a$.
- ii) Die Aussage ergibt sich aus i). $\square$

Bemerkung 2.3.21

Gilt $\sum_{i=j}^n h_i \leq a$, dann ist jeder Vektor des Typs $(\underbrace{0, \dots, 0, 1}_{(j-1)\text{-mal}}, d_{j+1}, \dots, d_n)$ mit $(d_{j+1}, \dots, d_n) \in \{0, 1\}^{n-j}$ ein Element von G_j .

Definition 2.3.22

Sei G_j eine Lösungsfamilie der linearen Ungleichung $h \underline{s} \leq a$.

$$F_j \hat{=} \{(\underline{d}, w(\underline{d})) \mid \underline{d} \in G_j \wedge w(\underline{d}) > 0\}$$

heißt Indikatorfamilie bezüglich G_j . j wird Familienindex genannt.

Ein Element $(\underline{d}, w(\underline{d})) \in F_j$ heißt Blockindikator.

$F = \bigcup_{j \in \mathbb{N}_n} F_j$ heißt Indikatormenge.

Bemerkung 2.3.23

$(\underline{s}, w(\underline{s})) \in F$ ist der Blockindikator eines Blockes, dessen zugeordneter Vektor $\underline{s}$ und dessen Wert $w(\underline{s}) > 0$ ist. Sei $(\underline{s}, w(\underline{s})) \in F_j$. Dann gilt für $\underline{s}$ $s_j = 1$ und $s_i = 0$ für $\forall i \in \{1, \dots, (j-1)\}$. Dies ist ein wichtiges Merkmal, das jedes Element von F_j besitzt.

Satz 2.3.24

Seien $i, j \in \mathbb{N}_n$.

$$i) |F_j| \leq |G_j|$$

$$ii) h_j = a \Rightarrow F_j = \emptyset$$

$$iii) F_n = \emptyset$$

$$iv) i \neq j \Rightarrow F_i \cap F_j = \emptyset$$

Beweis:

i) Dies ist eine unmittelbare Folgerung der Definition 2.3.22.

$$ii) h_j = a \Rightarrow G_j = \{\underline{e}_j\} \quad (\text{Lemma 2.3.16})$$

$$\Rightarrow F_j = \emptyset \quad (w(\underline{e}_j) = 0)$$

$$iii) G_n = \{\underline{e}_n\} \quad (\text{Folgerung 2.3.13ii})$$

$$\text{Daraus folgt, da\ss } F_n = \emptyset. \quad (w(\underline{e}_n) = 0)$$

iv) Dies ist eine unmittelbare Folgerung der Aussage

$$i \neq j \Rightarrow G_i \cap G_j = \emptyset. \quad \square \quad (\text{Folgerung 2.3.13iv})$$

Als Ergebnis der Phase 1 soll die Indikatormenge F in Form einer linearen Liste an die Phase 2 weitergeleitet werden. Dazu benötigen wir Ordnungsrelationen auf F_j bzw. auf F .

Definition 2.3.25

Sei F_j eine Indikatorfamilie.

$$RK_j \equiv \text{Max} \{w(\underline{s}) \mid (\underline{s}, w(\underline{s})) \in F_j\}$$

heißt Rang der Indikatorfamilie F_j .

Bezeichnung 2.3.26

Sei $\underline{s} = (s_1, \dots, s_n) \in \{0, 1\}^n$. Mit $\Sigma(\underline{s})$ wird die Summe $\sum_{i \in \mathbb{N}_n} 2^{i-1} s_i$ bezeichnet. Daraus folgt, daß

$$\underline{s} \neq \underline{s}' \iff \Sigma(\underline{s}) \neq \Sigma(\underline{s}') \quad \text{für } \forall \underline{s}, \underline{s}' \in \{0, 1\}^n.$$

Definition 2.3.27

Sei F_j eine Indikatorfamilie mit $|F_j| = p$.

$L_j = (F_j, R_f, \alpha_f)$ heißt F-Liste (kurz, FL) genau dann, wenn gilt:

$$i) F_j \neq \emptyset$$

ii) $R_f \subseteq F_j \times F_j$, angegeben durch:

$$(\underline{s}, w(\underline{s})) R_f (\underline{s}', w(\underline{s}')) \iff (w(\underline{s}) > w(\underline{s}')) \vee$$

$$(w(\underline{s}) = w(\underline{s}') \wedge |I_{\underline{s}}| < |I_{\underline{s}'}|) \vee$$

$$(w(\underline{s}) = w(\underline{s}') \wedge |I_{\underline{s}}| = |I_{\underline{s}'}| \wedge \underline{s} \succeq \underline{s}')$$

iii) $\alpha_f : F_j \xrightarrow{\text{bij}} \mathbb{N}_p$, wobei gilt:

$$\alpha_f(\underline{s}, w(\underline{s})) \leq \alpha_f(\underline{s}', w(\underline{s}')) \iff (\underline{s}, w(\underline{s})) R_f (\underline{s}', w(\underline{s}'))$$

$\alpha_f(\underline{s}, w(\underline{s})) \in \mathbb{N}_p$ heißt Index von $(\underline{s}, w(\underline{s}))$ in L_j .

Bemerkung 2.3.28

R_f ist eine lineare Ordnung auf F_j , d.h. R_f ist reflexiv, antisymmetrisch, transitiv, und alle Elemente von F_j sind miteinander vergleichbar.

Definition 2.3.29

Sei $L_j = (F_j, R_f, \alpha_f)$ eine F-Liste. Ein Element $(\underline{s}, w(\underline{s})) \in F_j$ heißt Vorstand von F_j genau dann wenn für $\forall (\underline{s}, w(\underline{s})) \in F_j$ folgendes gilt:

$$(\underline{s}, w(\underline{s})) R_f (\underline{s}, w(\underline{s})) \implies (\underline{s}, w(\underline{s})) = (\underline{s}, w(\underline{s})).$$

Der Vorstand von F_j wird mit r_j bezeichnet.

Satz 2.3.30

r_j ist der Vorstand von $F_j \iff \alpha_f(r_j) = 1$.

Beweis:

" $\implies$ " Annahme: $\alpha_f(r_j) \neq 1$, (d.h. $\alpha_f(r_j) > 1$)

$$\implies \text{Es gibt ein } (\underline{s}, w(\underline{s})) \neq r_j \text{ mit } \alpha_f(\underline{s}, w(\underline{s})) = 1$$

$$\implies (\underline{s}, w(\underline{s})) R_f r_j \wedge (\underline{s}, w(\underline{s})) \neq r_j$$

$$\implies r_j \text{ ist nicht Vorstand von } F_j. \text{ Widerspruch.}$$

" $\Leftarrow$ " Annahme: r_j ist nicht Vorstand von F_j .

$$\Rightarrow \exists (\underline{s}, w(\underline{s})) \in F_j : ((\underline{s}, w(\underline{s})) R_f r_j \wedge (\underline{s}, w(\underline{s})) \neq r_j)$$

$$\Rightarrow \alpha_f((\underline{s}, w(\underline{s}))) < \alpha_f(r_j).$$

Im Widerspruch zu der Voraussetzung daß $\alpha_f(r_j) = 1$. $\square$

Bezeichnung 2.3.31

Sei k die Anzahl der F -Listen, die aus einer Indikatormenge F gebildet werden. Aus Def. 2.3.27 folgt $k \leq n$, wobei n die Anzahl der Indikatorfamilien von F angibt. $LT = \{L_j \mid j \in \mathbb{N}_n \wedge L_j = (F_j, R_f, \alpha_f)\}$ bezeichnet die Menge aller FL von F .

Definition 2.3.32

Sei F eine Indikatormenge. Seien F_i, F_j Indikatorfamilien von F . Seien $L_i = (F_i, R_f, \alpha_f)$, $L_j = (F_j, R_f, \alpha_f)$. Sei $(\underline{x}, w(\underline{x}))$ der Vorstand von F_i und $(\underline{y}, w(\underline{y}))$ der Vorstand von F_j .

Eine L-Liste ist ein Tripel $LL = (LT, R_l, \alpha_l)$, wobei gilt:

i) $LT = \{L_j \mid j \in \mathbb{N}_n \wedge L_j = (F_j, R_f, \alpha_f)\}$ wobei $|LT| = k$.

ii) $R_l \subseteq LT \times LT$, angegeben durch:

$$\begin{aligned} L_i R_l L_j &\iff (RK_i > RK_j) \vee \\ &\quad (RK_i = RK_j \wedge |F_i| < |F_j|) \vee \\ &\quad (RK_i = RK_j \wedge |F_i| = |F_j| \wedge \underline{x} \geq \underline{y}) \end{aligned}$$

iii) $\alpha_l: LT \xrightarrow{\text{bij}} \mathbb{N}_k$ wobei gilt:

$$\alpha_l(L_i) \leq \alpha_l(L_j) \iff L_i R_l L_j$$

$\alpha_l(L_i)$ bzw. $\alpha_l(L_j)$ heit Index von L_i bzw. von L_j in LT .

Bemerkung 2.3.33

R_l ist eine lineare Ordnungsrelation auf LT .

Satz 2.3.34

i) Es gibt genau eine $L_m \in LT$, soda fr $\forall L_i \in LT$ folgendes gilt:

$$L_i R_l L_m \iff L_i = L_m$$

ii) $L_i = L_m \iff \alpha_l(L_i) = 1$.

Beweis:

i) L_m ist das maximale Element von (LT, R_1) .

ii) Wie Satz 2.3.30. $\square$

Definition 2.3.35

Sei F eine Indikatormenge mit $|F| = m$. Seien F_i, F_j Indikatorfamilien von F . Seien $L_i = (F_i, R_f, \alpha_f)$, $L_j = (F_j, R_f, \alpha_f)$. Sei $(\underline{x}, w(\underline{x})) \in F_i$ und $(\underline{y}, w(\underline{y})) \in F_j$.

Eine E-Liste ist ein Tripel $EL = (F, R_e, \alpha_e)$, wobei gilt:

i) $R_e \subseteq F \times F$, angegeben durch:

$$(\underline{x}, w(\underline{x})) R_e (\underline{y}, w(\underline{y})) \iff (\alpha_1(L_i) < \alpha_1(L_j)) \vee \\ (\alpha_1(L_i) = \alpha_1(L_j) \wedge \alpha_f((\underline{x}, w(\underline{x}))) \leq \alpha_f((\underline{y}, w(\underline{y}))))$$

ii) $\alpha_e: F \xrightarrow{\text{bij}} \mathbb{N}_m$, angegeben durch:

$$\alpha_e((\underline{x}, w(\underline{x}))) \leq \alpha_e((\underline{y}, w(\underline{y}))) \iff (\underline{x}, w(\underline{x})) R_e (\underline{y}, w(\underline{y}))$$

$\alpha_e((\underline{x}, w(\underline{x})))$ bzw. $\alpha_e((\underline{y}, w(\underline{y})))$ heit Listennummer von $(\underline{x}, w(\underline{x}))$ bzw. von $(\underline{y}, w(\underline{y}))$ in EL .

Bemerkung 2.3.36

R_e ist eine lineare Ordnung auf F . Jedem Element in F wird genau eine Listennummer $i \in \mathbb{N}_m$ zugeordnet. Sei i die Listennummer von $(\underline{s}, w(\underline{s}))$, dann wird statt $(\underline{s}, w(\underline{s}))$ $(\underline{s}_i, w_i)$ geschrieben. Wir werden knftig F mit $|F| = m$ als eine indizierte Menge betrachten:

$$F \cong \left\{ (\underline{s}_i, w_i) \mid i \in \mathbb{N}_m \right\}.$$

Im folgenden werden wir eine rekursive Prozedur zur Auffindung aller Indikatorfamilien angeben. Der Algorithmus beruht auf den Lemmata 2.3.16 - 2.3.20.

Algorithmus 2.3.37

Name: "Prozedur F"

Voraussetzung: $h = (h_1, \dots, h_n)$ ist die durch VOR induzierte Knotengewichtung.

a ist die Gewichtsgrenze, wobei $a \geq h_1 \geq \dots \geq h_n > 0$ und $\sum_{i \in \mathbb{N}_n} h_i > a$.

$(f_{kl})_{k,l \in \mathbb{N}_n}$ ist die Matrix der Kantengewichte.

Sei $\underline{d} = (d_1, \dots, d_n)$ eine Lösung von $hs \leq a$, dann wird $(d_1, \dots, d_k)$ mit $k < n$ als eine partielle Lösung bezeichnet.

Startbedingung: $F = \emptyset$, $m = 1$.

Iteration:

Wiederhole für $j = m, m+1, \dots, n$ die folgenden Schritte:

- 1) Für jedes j werden die Bedingungen der Fälle 1, ..., 5 getestet. Ist die Bedingung eines bestimmten Falls i ($i \in \mathbb{N}_5$) erfüllt, dann wird gemäß der Aussage des entsprechenden Lemmas (2.3.16-2.3.20) eine partielle Lösung festgelegt oder eine Teilmenge von G_j aufgefunden. Wenn eine partielle Lösung $(d_1, \dots, d_k)$ vorliegt, $m := k+1$ und dieselbe Prozedur wird wieder aufgerufen.
- 2) Für jedes $\underline{d} \in G_j$ wird $w(\underline{d})$ berechnet. Anschließend wird F_j gebildet.

Ergebnis: $F = \bigcup_{j \in \mathbb{N}_n} F_j$.

Bemerkung 2.3.38

Jeder (rekursive) Prozeduraufruf wird als eine Iteration bezeichnet. Bei jeder Iteration wird der Test der Bedingungen für $j = m, m+1, \dots, n$ durchgeführt. Der Rechenaufwand zur Auffindung eines Blockindicators bei einer Iteration wird als Rechenschritt bezeichnet.

Satz 2.3.39

Die Prozedur F benötigt höchstens $\frac{1}{2}(n^2 - n)$ Iterationen zur Auffindung der Indikatormenge F . Jede Iteration, bei der die Teilmenge einer Indikatorfamilie aufgefunden wird, benötigt $O(\max(n, 2^{n-m}))$ Rechenschritte.

Beweis:

Teil 1 Wenn der Fall 1 oder der Fall 5 für ein bestimmtes j vorliegt, wird die Prozedur F nicht rekursiv aufgerufen.

In allen anderen Fällen besteht die Möglichkeit, daß die Prozedur so lange rekursiv aufgerufen werden muß, bis ein m erreicht wird, das der Bedingung des Falls 5 genügt. Für eine bestimmte Lösungsfamilie G_j wird die Anzahl der rekursiven Prozeduraufrufe auf $n-j$ beschränkt.

Deshalb bleibt die Gesamtzahl der Iterationen $\sum_{j \in \mathbb{N}_n} (n-j) = \frac{1}{2}(n^2 - n)$.

Teil 2 Die Teilmenge einer Indikatorfamilie wird aufgefunden, wenn die Bedingung von Fall 1, von Fall 3 oder von Fall 5 erfüllt wird. Fall 1 und Fall 3 benötigen dazu $O(n)$ Rechenschritte. Im Falle 5 ist die Anzahl der Rechenschritte $O(2^{n-m})$. Bei jeder Iteration wird die Bedingung von Fall 5 höchstens einmal getestet. $\square$

Bemerkung 2.3.40

Aus Satz 2.3.39 folgt, daß Probleme, bei denen die Bedingung von Fall 5 bei relativ kleinem m (im Verhältniss zu n) erfüllt wird, sehr großen Rechenaufwand erfordern (in diesem Falle gilt $2^{n-m} \approx 2^n$). Für großes n wird die E-Liste einen solchen Umfang erreichen, daß sie kaum für den binären Verzweigungsprozeß der Phase 2 brauchbar sein kann. Dies ist ein großer Nachteil der Prozedur F, der dadurch entsteht, daß der Algorithmus auf den Aussagen der Lemmata 2.3.16 - 2.3.20 beruht, die ausschließlich kombinatorische Eigenschaften des Problems berücksichtigen.

Algorithmus 2.3.41

Name: Prozedur SORT

Voraussetzung: $F = \bigcup_{j \in \mathbb{N}_n} F_j$ ist gegeben.

Startbedingung: EL ist leer.

Iteration:

- 1) Sortiere gemäß der Ordnungsrelation R_f (Def. 2.3.27) die Elemente jeder nicht leeren Indikatorfamilie F_j und bilde die entsprechende F-Liste L_j . Wiederhole die Sortierung für $\forall j \in \mathbb{N}_n$.
- 2) Sortiere gemäß der Ordnungsrelation R_l (Def. 2.3.32) die Elemente der Menge LT und bilde anschließend gemäß der Ordnungsrelation R_e (Def. 2.3.35) die E-Liste.

Ergebnis: $EL = (F, R_e, \alpha_e)$.

Bemerkung 2.3.42

Effiziente Sortieralgorithmen, die auf dem Prinzip des paarweisen Vergleichens beruhen, können für die Prozedur SORT eingesetzt werden. Da solche Algorithmen weitgehend bekannt und an verschiedenen Literaturstellen (z.B. /93/, Kap.3)

in allen Einzelheiten angegeben sind, verzichten wir hier auf ihre detaillierte Beschreibung. Wir bezeichnen eine Operation, welche zwei Elemente einer Menge miteinander vergleicht, als einen Rechenschritt der Prozedur SORT.

Satz 2.3.43

Sei S eine n -elementige Menge und R eine lineare Ordnungsrelation auf S . Jeder Algorithmus, der die Elemente in S paarweise miteinander vergleicht und sie nach R ordnet, benötigt $O(n \log_2 n)$ Vergleichsoperationen.

Beweis: Der Beweis dieses Satzes ist z.B. in /93/ angegeben. $\square$

Satz 2.3.44

Prozedur SORT bricht nach $O(\max(\sum_{j \in \mathbb{N}_n} |F_j| \log_2 |F_j|, n \log_2 n))$ Rechenschritten ab.

Beweis:

Zur Sortierung der Elemente in F_j werden nach Satz 2.3.43 $O(|F_j| \log_2 |F_j|)$ Rechenschritte gebraucht. Um die Elemente in allen Indikatorfamilien zu sortieren, benötigt man $O(\sum_{j \in \mathbb{N}_n} |F_j| \log_2 |F_j|)$ Rechenschritte.

$O(n \log_2 n)$ Rechenschritte werden benötigt, um die F -Listen in LT gemäß R_j zu ordnen. Danach braucht man keine Vergleichsoperationen mehr auszuführen, da die Elemente von F bereits linear geordnet sind und jedes Element eine Listennummer erhält, die seinem Platz in der linearen Ordnung entspricht. $\square$

Beispiel 2.3.45

Ein Netz $NG = (V, E, g, f)$ sei folgendermaßen angegeben (Bild 2.3.1):

$$V = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$$

$$g = (7, 1, 6, 4, 5, 2, 6, 3, 4, 1, 8, 5)$$

$$E = \{\{1,2\}, \{2,3\}, \{2,4\}, \{2,7\}, \{3,4\}, \{3,7\}, \{4,5\}, \{5,6\}, \\ \{5,8\}, \{5,11\}, \{6,7\}, \{7,8\}, \{8,9\}, \{9,10\}, \{10,11\}, \{11,12\}\}$$

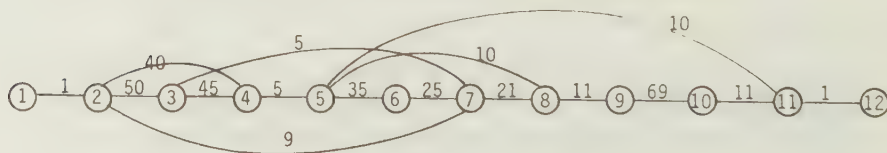


Bild 2.3.1

$f = (f_{kl})_{k,l \in W_n}$, angegeben durch die folgende Matrix:

	1	2	3	4	5	6	7	8	9	10	11	12
1	0	1	0	0	0	0	0	0	0	0	0	0
2	1	0	50	40	0	0	9	0	0	0	0	0
3	0	50	0	45	0	0	5	0	0	0	0	0
4	0	40	45	0	5	0	0	0	0	0	0	0
5	0	0	0	5	0	35	0	10	0	0	10	0
6	0	0	0	0	35	0	25	0	0	0	0	0
7	0	9	5	0	0	25	0	21	0	0	0	0
8	0	0	0	0	10	0	21	0	11	0	0	0
9	0	0	0	0	0	0	0	11	0	69	0	0
10	0	0	0	0	0	0	0	0	69	0	11	0
11	0	0	0	0	10	0	0	0	0	11	0	1
12	0	0	0	0	0	0	0	0	0	0	1	0

Sei $a = 8$. Sei $\underline{x} = (x_1, \dots, x_{12})$.

$$g\underline{x} = 7x_1 + x_2 + 6x_3 + 4x_4 + 5x_5 + 2x_6 + 6x_7 + 3x_8 + 4x_9 + x_{10} + 8x_{11} + 5x_{12} \leq 8,$$

Statt $g\underline{x} \leq 8$ betrachten wir nun die lineare Ungleichung $h\underline{s} \leq 8$:

$$h\underline{s} = 8s_1 + 7s_2 + 6s_3 + 6s_4 + 5s_5 + 5s_6 + 4s_7 + 4s_8 + 3s_9 + 2s_{10} + s_{11} + s_{12} \leq 8,$$

wobei folgendes gilt (Def. 2.3.3):

$$\begin{array}{ll} h_1 = g_{11} = 8 & s_1 = x_{11} \\ h_2 = g_1 = 7 & s_2 = x_1 \\ h_3 = g_3 = 6 & s_3 = x_3 \\ h_4 = g_7 = 6 & s_4 = x_7 \\ h_5 = g_5 = 5 & s_5 = x_5 \\ h_6 = g_{12} = 5 & s_6 = x_{12} \\ h_7 = g_4 = 4 & s_7 = x_4 \\ h_8 = g_9 = 4 & s_8 = x_9 \end{array}$$

$$\begin{array}{ll}
 h_9 = g_8 = 3 & s_9 = x_8 \\
 h_{10} = g_6 = 2 & s_{10} = x_6 \\
 h_{11} = g_2 = 1 & s_{11} = x_2 \\
 h_{12} = g_{10} = 1 & s_{12} = x_{10}
 \end{array}$$

Die Lösungsmenge G lässt sich in 12 Lösungsfamilien G_j , $j \in \mathbb{N}_{12}$, aufteilen (Def. 2.3.12 und Lemma 2.3.16 - 2.3.20):

G_1 :	100000000000	G_6 :	000001000010 000001000001 000001000000	G_9 :	000000001111 000000001110 000000001101 000000001100 000000001011 000000001010 000000001001 000000001000
G_2 :	010000000010 010000000001 010000000000	G_7 :	000000110000 000000101010 000000101001 000000101000 000000100111 000000100110 000000100101 000000100100	G_{10} :	000000000111 000000000110 000000000101 000000000100
G_3 :	001000000100 001000000011 001000000010 001000000001 001000000000			G_{11} :	000000000011 000000000010
G_4 :	000100000100 000100000011 000100000010 000100000001 000100000000			G_{12} :	000000000001
G_5 :	000010001000 000010000110 000010000101 000010000100 000010000011 000010000010 000010000001 000010000000	G_8 :	000000011010 000000011001 000000011000 000000010111 000000010110 000000010101 000000010100 000000010011 000000010010		
G_6 :	000001001000 000001000110 000001000101 000001000100 000001000011				

Die Prozedur F und die Prozedur SORT werden nacheinander ausgeführt.
Daraus ergibt sich die E-Liste:

i	L_j	α_f	α_q	s_i	w_i	$RK(\alpha_q)$
1	L_8	1	1	0 0 0 0 0 0 0 1 1 0 0 1	80	80
2		2		0 0 0 0 0 0 0 1 0 0 0 1	69	
3		3		0 0 0 0 0 0 0 1 0 1 0 1	69	
4		4		0 0 0 0 0 0 0 1 0 0 1 1	69	
5		5		0 0 0 0 0 0 0 1 0 1 1 1	69	
6		6		0 0 0 0 0 0 0 1 1 0 0 0	11	
7		7		0 0 0 0 0 0 0 1 1 0 1 0	11	
8	L_3	1	2	0 0 1 0 0 0 0 0 0 0 1 0	50	50
9		2		0 0 1 0 0 0 0 0 0 0 1 1	50	
10	L_7	1	3	0 0 0 0 0 0 1 0 0 0 1 0	40	40
11		2		0 0 0 0 0 0 1 0 1 0 1 0	40	
12		3		0 0 0 0 0 0 1 0 0 1 1 0	40	
13		4		0 0 0 0 0 0 1 0 0 0 1 1	40	
14		5		0 0 0 0 0 0 1 0 0 1 1 1	40	
15	L_5	1	4	0 0 0 0 1 0 0 0 0 1 0 0	35	35
16		2		0 0 0 0 1 0 0 0 0 1 1 0	35	
17		3		0 0 0 0 1 0 0 0 0 1 0 1	35	
18		4		0 0 0 0 1 0 0 0 1 0 0 0	10	
19	L_4	1	5	0 0 0 1 0 0 0 0 0 1 0 0	25	25
20		2		0 0 0 1 0 0 0 0 0 0 1 0	9	
21		3		0 0 0 1 0 0 0 0 0 0 1 1	9	
22	L_2	1	6	0 1 0 0 0 0 0 0 0 0 1 0	1	1

2.4 Der binäre Verzweigungsprozeß

Ausgangspunkt des in diesem Abschnitt zu behandelnden Themas ist die Indikatormenge F und die E -Liste, die eine lineare Ordnung auf F darstellt. Wir betrachten zunächst das folgende Problem, das als Problem der Phase 2 (kurz, PH2) bezeichnet wird:

Problem PH2

Parameter: F , wobei $|F| = m > 0$.

Gesucht: $\hat{Z}_p$ und $\hat{U}_p$, für die folgendes gilt:

$$\hat{Z}_p = \text{Max} \left\{ W(U_p) \mid U_p \in H_p \wedge W(U_p) = \bigwedge_{(\underline{s}_i, w_i) \in U_p} w_i \right\}$$

$$\hat{U}_p = \{ U_p \mid U_p \in H_p \wedge W(U_p) = \hat{Z}_p \}, \text{ wobei}$$

$$H_p = \left\{ U_p \mid U_p \subseteq 2^F \wedge \forall (\underline{s}_i, w_i), (\underline{s}_j, w_j) \in U_p : (\underline{s}_i \neq \underline{s}_j \Rightarrow \underline{s}_i \wedge \underline{s}_j = \underline{0}) \right\}.$$

$\hat{U}_p \times \{\hat{Z}_p\}$ heißt Lösungsmenge von PH2. Jedes Element $(U_p, \hat{Z}_p) \in \hat{U}_p \times \{\hat{Z}_p\}$ ist eine Lösung von PH2.

Satz 2.4.1

Das Teilproblem Q_2 ist gelöst wenn die Lösungsmenge von PH2 bekannt ist.

Beweis:

Die Parameter C_1 und D_1 des Teilproblems Q_2 (Abschn. 2.2) läßt sich aus F folgendermaßen ableiten:

Sei $(\underline{s}_i, w_i) \in F$. Aus $\underline{s}_i$ wird der charakteristische Vektor $\underline{x}$ gebildet, dessen zugeordneter Vektor $\underline{s}_i$ ist (Def. 2.3.3) und bei dem gilt: $w_i = w(\underline{x})$ (Lemma 2.3.6). Für jeden Blockindikator $(\underline{s}_i, w_i)$ gibt es genau ein $\underline{x} \in C_1$, dessen Wert $w(\underline{x}) \in D_1$ ist. Umgekehrt läßt sich aus einem $(\underline{x}, w(\underline{x})) \in C_1 \times D_1$ genau ein $(\underline{s}_i, w_i) \in F$ bilden.

Es gibt also eine bijektive Abbildung $\varphi: F \xrightarrow{\text{bij}} C_1$, für die gilt:

$$\varphi((\underline{s}_i, w_i)) = \underline{x} \iff \underline{s}_i \text{ ist der zugeordnete Vektor von } \underline{x} \wedge w_i = w(\underline{x}) \in D_1.$$

Berücksichtigt man die Aussage iii) des Satzes 2.3.7, so ist die Existenz der bijektiven Abbildung $\varphi': H_p \xrightarrow{\text{bij}} H'$ leicht zu erkennen, für die gilt:

$$\varphi'(U_p) = U' \iff W(U_p) = W(U') \wedge U_p = \{(\underline{s}_i, w_i) \mid \underline{s}_i = \text{zugeord. Vektor von } \underline{x} \in U'\}.$$

Daraus folgt: $\hat{Z} = \hat{Z}_p$.

Aus den Definitionen von $\hat{U}'$ und $\hat{U}_p$ ist es leicht einzusehen, daß es eine bijektive Abbildung $\varphi'': \hat{U}_p \rightarrow \hat{U}'$ gibt, für die gilt:

$$\varphi''(U_p) = U' \iff U_p = \{(\underline{s}_i, w_i) \mid \underline{s}_i = \text{zugeord. Vektor von } \underline{x} \in U'\} \wedge W(U_p) = W(U') = \hat{Z}.$$

Aus $\hat{U}_p \times \{\hat{Z}_p\}$ kann die Lösungsmenge des Teilproblems Q_2 abgeleitet werden. $\square$

Definition 2.4.2

Die Abbildung

$$y_{U_p}: F \rightarrow \{0,1\}, \text{ angegeben durch}$$

$$y_{U_p}((\underline{s}_i, w_i)) = \begin{cases} 1, & \text{wenn } (\underline{s}_i, w_i) \in U_p, \\ 0, & \text{wenn } (\underline{s}_i, w_i) \in F \setminus U_p, \end{cases}$$

heißt charakteristische Funktion von U_p bezüglich F .

Bemerkung 2.4.3

y_{U_p} ist ein Element aus dem m -dimensionalen Vektorraum über $GF(2)$. Mit $\underline{y} = (y_1, \dots, y_m) \in \{0,1\}^m$ wird der charakteristische Vektor von U_p bezeichnet, dabei wird $y_i = y_{U_p}((\underline{s}_i, w_i))$ die i -te Komponente des Vektors $\underline{y}$ genannt. Mit $y_i \underline{s}_i$ wird der Vektor $(y_i s_{i1}, \dots, y_i s_{ij}, \dots, y_i s_{in})$ bezeichnet. Daraus folgt, daß $y_i \underline{s}_i = \underline{0}$, wenn $y_i = 0$, und $y_i \underline{s}_i = \underline{s}_i$, wenn $y_i = 1$.

Lemma 2.4.4

Sei $\underline{y} = (y_1, \dots, y_m)$ der charakteristische Vektor von U_p .

$$\text{i) } \sum_{(\underline{s}_i, w_i) \in U_p} w_i = \sum_{i \in I_{\underline{y}}} w_i$$

$$\text{ii) } \bigvee_{(\underline{s}_i, w_i) \in U_p} \underline{s}_i = \bigvee_{i \in I_{\underline{y}}} \underline{s}_i$$

$$\begin{aligned} \text{iii) } & \forall (\underline{s}_i, w_i), (\underline{s}_j, w_j) \in U_p: (\underline{s}_i \neq \underline{s}_j \implies \underline{s}_i \wedge \underline{s}_j = \underline{0}) \\ & \iff \forall i, j \in N_m: (i \neq j \implies y_i \underline{s}_i \wedge y_j \underline{s}_j = \underline{0}) \end{aligned}$$

Beweis:

i), ii) und iii) sind unmittelbare Folgerungen der Definition 2.4.2 und der Bemerkung 2.4.3. $\square$

Bemerkung 2.4.5

$W(\underline{y}) \equiv \sum_{i \in I_{\underline{y}}} w_i$ wird als Wert des Vektors $\underline{y}$ bezeichnet.

Um die mathematische Handhabung zu vereinfachen, werden wir künftig statt U_p dessen charakteristischen Vektor $\underline{y}$ für die Behandlung des Problems PH2 verwenden. Dazu muß PH2 umformuliert werden:

Problem P2

Parameter: F , wobei $|F| = m > 0$.

Gesucht: $\hat{Z}_q$ und $\hat{\theta}$, für die folgendes gilt:

$$\hat{Z}_q = \text{Max} \{ W(\underline{y}) \mid \underline{y} \in \theta \}$$

$$\hat{\theta} = \{ \underline{y} \mid \underline{y} \in \theta \wedge W(\underline{y}) = \hat{Z}_q \}$$

$$\theta = \{ \underline{y} \mid \underline{y} \in \{0,1\}^m \wedge \forall i, j \in \mathbb{N}_m : (i \neq j \Rightarrow y_i \leq_i \wedge y_j \leq_j = 0) \}.$$

$\hat{\theta} \times \{ \hat{Z}_q \}$ heißt Lösungsmenge von P2. Jedes Element $(\underline{y}, \hat{Z}_q) \in \hat{\theta} \times \{ \hat{Z}_q \}$ ist eine Lösung von P2.

Satz 2.4.6

Problem PH2 ist gelöst, wenn die Lösungsmenge von P2 bekannt ist.

Beweis:

Betrachtet man die Aussage iii) des Lemmas 2.4.4 und die Definition 2.4.2, dann ist es leicht einzusehen, daß eine bijektive Abbildung $\mathcal{J}: \theta \longrightarrow H_p$ existiert, für die gilt:

$$\mathcal{J}(\underline{y}) = U_p \iff \underline{y} = \text{der charakt. Vektor von } U_p \wedge W(\underline{y}) = W(U_p).$$

Aus der Aussage i) des Lemmas 2.4.4 folgt $\hat{Z}_q = \hat{Z}_p$.

Damit ist die Existenz einer bijektiven Abbildung $\mathcal{J}: \hat{\theta} \longrightarrow \hat{U}_p$ nachgewiesen, für die gilt:

$$r'(y) = U_p \iff y = \text{der charakt. Vektor von } U_p \wedge W(y) = W(U_p) = \hat{Z}_q.$$

Aus der Lösungsmenge von P2 läßt sich also die Lösungsmenge von PH2 ableiten. $\square$

Definition 2.4.7

Ein vollständiger Binärbaum $T = (P, A)$ der Höhe m heißt binärer Verzweigungsbaum genau dann, wenn gilt:

- i) $P \cong \{ \underline{u} \mid \underline{u} \in \{0,1\}^i \text{ für } \forall i \in \{0,1,\dots,m\} \}$ ist die Knotenmenge von T .
- ii) $\emptyset \cong () \cong \{0,1\}^0$ ist die Wurzel von T .
- iii) $A \subseteq P \times P \setminus \Delta_P$ ist die Kantenmenge, angegeben durch:

$$(\underline{u}, \underline{v}) \in A \iff (\underline{u} = (u_1, \dots, u_i) \implies \underline{v} = (u_1, \dots, u_i, 0) \vee \underline{v} = (u_1, \dots, u_i, 1)).$$

Sei $\underline{u} = (u_1, \dots, u_i) \in P$. $(u_1, \dots, u_i, 0)$ heißt Nein-Sohn von $\underline{u}$. $(u_1, \dots, u_i, 1)$ heißt Ja-Sohn von $\underline{u}$.

Die Höhe des Knotens $\underline{u}$ wird mit $|\underline{u}|$ bezeichnet.

Folgerung 2.4.8

- i) $\underline{u} \in \{0,1\}^i \implies |\underline{u}| = i$.
- ii) Sei $(\underline{u}, \underline{v}) \in A$. Sei $|\underline{v}| = j \geq 1$.
 $\underline{v}$ ist der Nein-Sohn von $\underline{u} \implies I_{\underline{v}} = I_{\underline{u}}$.
 $\underline{v}$ ist der Ja-Sohn von $\underline{u} \implies I_{\underline{v}} = I_{\underline{u}} \cup \{j\}$.

Bemerkung 2.4.9

Einige mit dem Binärbaum zusammenhängende Begriffe und Eigenschaften, die in dieser Arbeit verwendet werden, sind in Anhang B angegeben.

Bezeichnung 2.4.10

Mit $P_j = \{ \underline{u} \mid \underline{u} \in \{0,1\}^j \}$ wird die Menge aller Knoten der Höhe $j \leq m$ bezeichnet.
 $P_m = \{ \underline{u} \mid \underline{u} \in \{0,1\}^m \}$ bezeichnet die Menge aller Blätter von T .

Satz 2.4.11

- i) $\underline{u} \in P_j \implies |\underline{u}| = j$
- ii) $|P_j| = 2^j$

$$\text{iii) } P = \bigcup_{i \in \{0,1,-,m\}} P_i$$

$$\text{iv) } \bigcap_{i \in \{0,1,-,m\}} P_i = \emptyset$$

$$\text{v) } |P| = 2^{m+1} - 1.$$

Beweis: Offensichtlich. $\square$

Definition 2.4.12

Sei F eine Indikatormenge mit $|F|=m$. Sei $k \in \mathbb{N}$ die Anzahl der F -Listen von F . Sei $EL = (F, R_e, \alpha_e)$ die E -Liste von F . Eine binäre Verzweigungsstruktur bezüglich EL ist ein Siebentupel $VST = (P, A, F, \mathcal{P}, \omega, \varphi, \underline{\mathcal{E}})$, wobei gilt:

- i) (P, A) ist ein binärer Verzweigungsbaum der Höhe m .
- ii) $\mathcal{P}, \omega, \varphi, \underline{\mathcal{E}}$ sind Knotenbewertungsfunktionen, die folgendermaßen angegeben werden:

$\mathcal{P}: P \longrightarrow FU\{(\underline{0}, 0)\}$, definiert durch:

$$\mathcal{P}(\underline{u}) = \begin{cases} (\underline{s}_i, w_i) & \text{wenn } |\underline{u}| = i \in \mathbb{N}_m. \\ (\underline{0}, 0) & \text{wenn } |\underline{u}| = 0 \text{ (d.h. } \underline{u} = \underline{0}). \end{cases}$$

$\mathcal{P}_l(\underline{u})$ bezeichnet die linke Komponente von $\mathcal{P}(\underline{u})$, d.h. $\mathcal{P}_l(\underline{u}) = \underline{s}_i$ bzw. $\mathcal{P}_l(\underline{u}) = \underline{0}$.

$\mathcal{P}_r(\underline{u})$ bezeichnet die rechte Komponente von $\mathcal{P}(\underline{u})$, d.h. $\mathcal{P}_r(\underline{u}) = w_i$ bzw. $\mathcal{P}_r(\underline{u}) = 0$.

$\omega: P \longrightarrow \mathbb{N}_0$, definiert durch:

$$\omega(\underline{v}) = \begin{cases} 0 & \text{wenn } \underline{v} = \underline{0}. \\ \omega(\underline{u}) & \text{wenn } \underline{v} = \text{Nein-Sohn von } \underline{u}. \\ \omega(\underline{u}) + \mathcal{P}_r(\underline{v}) & \text{wenn } \underline{v} = \text{Ja-Sohn von } \underline{u}. \end{cases}$$

$\varphi: P \longrightarrow \{0,1\}^n$, definiert durch:

$$\varphi(\underline{v}) = \begin{cases} \underline{0} & \text{wenn } \underline{v} = \underline{0}. \\ \varphi(\underline{u}) & \text{wenn } \underline{v} = \text{Nein-Sohn von } \underline{u}. \\ \varphi(\underline{u}) \vee \mathcal{P}_l(\underline{v}) & \text{wenn } \underline{v} = \text{Ja-Sohn von } \underline{u}. \end{cases}$$

(n ist die Dimension des Vektors $\underline{s}_i$)

$\Phi: P \rightarrow N_0$, definiert durch:

$$\Phi(\underline{u}) = \begin{cases} \sum_{i \in \mathbb{N}_k} RK_i, & \text{wenn } \underline{u} = \underline{\emptyset}. \\ \sum_{i=\alpha_1(L_j)+1}^k RK_i + \rho_r(\underline{v}), & \text{wenn } \underline{u} = \text{Nein-Sohn } \underline{u} \wedge (\underline{u}, \underline{v}) \in A \\ & \wedge \rho(\underline{v}), \rho(\underline{u}) \notin L_j \wedge \alpha_1(L_j) < k. \\ \sum_{i=\alpha_1(L_j)+1}^k RK_i, & \text{wenn } \underline{u} = \text{Ja-Sohn } \wedge \rho(\underline{u}) \notin L_j \wedge \alpha_1(L_j) < k \\ & \text{oder } (\underline{u}, \underline{v}) \in A \wedge \rho(\underline{u}) \notin L_j \wedge \alpha_1(L_j) < k \\ & \wedge \rho(\underline{v}) \notin L_j \wedge \alpha_1(L_j) = \alpha_1(L_j) + 1. \\ 0, & \text{wenn } \rho(\underline{u}) \notin L_j \wedge \alpha_1(L_j) = k. \end{cases}$$

Dabei ist RK_i , $i \in \mathbb{N}_k$, der Rang der F-Liste $L_j \in LT$ mit $\alpha_1(L_j) = i$.

$\rho(\underline{u}) \notin L_j$ bedeutet, daß $\rho(\underline{u})$ ein Element der Liste L_j ist.

Satz 2.4.13

Sei $VST = (P, A, F, \rho, \omega, \varphi, \Phi)$ eine binäre Verzweigungsstruktur bezüglich $EL = (F, R_e, \alpha_e)$. Sei $\underline{v} \in P$ mit $\underline{v} \neq \underline{\emptyset}$.

$$i) \quad \omega(\underline{v}) = \sum_{i \in I_{\underline{v}}} w_i$$

$$ii) \quad \underline{v} \text{ ist ein Nachfolger von } \underline{u} \in P \Rightarrow \omega(\underline{u}) \leq \omega(\underline{v})$$

$$iii) \quad \varphi(\underline{v}) = \bigvee_{i \in \mathbb{N}_k} v_i s_i$$

$$iv) \quad \underline{v} \text{ ist ein Nachfolger von } \underline{u} \in P \Rightarrow |I_{\varphi(\underline{u})}| \leq |I_{\varphi(\underline{v})}|$$

$$v) \quad \underline{v} \text{ ist ein Nachfolger von } \underline{u} \in P \Rightarrow \Phi(\underline{v}) \leq \Phi(\underline{u})$$

$$vi) \quad \underline{v} \text{ ist ein Blatt} \Rightarrow \Phi(\underline{v}) = 0.$$

Beweis:

i) Da $\underline{v} \neq \underline{\emptyset}$, so hat $\underline{v}$ mindestens einen Vorgänger. Sei $\underline{v} = (v_1, \dots, v_j)$.

Vollständige Induktion über j :

$j = 1$: $\underline{\emptyset}$ ist Vater von $\underline{v}$. Aus Def. 2.4.12 folgt:

$$\omega(\underline{v}) = \omega(\underline{\emptyset}) = 0, \quad \text{wenn } \underline{v} = \text{Nein-Sohn von } \underline{\emptyset}.$$

$$\omega(\underline{v}) = 0 + w_1 = w_1, \quad \text{wenn } \underline{v} = \text{Ja-Sohn von } \underline{\emptyset}.$$

Sei $\omega(\underline{v}) = \sum_{i \in I_{\underline{v}}} w_i$ für $j = k < m$.

Sei $j = k+1$. Sei $\underline{v}' = (v_1, \dots, v_{k+1})$.

$\underline{v}' = \text{Nein-Sohn von } \underline{v} \Rightarrow \omega(\underline{v}') = \sum_{i \in I_{\underline{v}'}} w_i \quad (I_{\underline{v}} = I_{\underline{v}'})$

$\underline{v} = \text{Ja-Sohn von } \underline{v} \Rightarrow \omega(\underline{v}') = \omega(\underline{v}) + \rho_r(\underline{v}') = \sum_{i \in I_{\underline{v}}} w_i + w_{k+1} = \sum_{i \in I_{\underline{v}'}} w_i$.

ii) Ergibt sich aus der Aussage i).

iii) Der Beweis kann wie i) durchgeführt werden.

iv) Die Aussage gilt offensichtlich für $\underline{u} = \underline{v}$.

Falls $(\underline{u}, \underline{v}) \in A$, dann folgt unmittelbar aus iii) :

$$|I_{\varphi(\underline{v})}| \geq |I_{\varphi(\underline{u})}|$$

Sei $(\underline{u}, \underline{u}^1, \dots, \underline{u}^s, \underline{v})$ eine Bahn. Dann gilt offensichtlich:

$$|I_{\varphi(\underline{v})}| \geq |I_{\varphi(\underline{u}^s)}| \geq \dots \geq |I_{\varphi(\underline{u}^1)}| \geq |I_{\varphi(\underline{u})}|.$$

v) Die Aussage ergibt sich aus Def. 2.3.35 und Def. 2.4.12.

vi) Die Aussage ergibt sich aus Def. 2.4.12. $\square$

Definition 2.4.14

Sei $VST = (P, A, F, \rho, \omega, \varphi, \mathcal{E})$ eine binäre Verzweigungsstruktur bezüglich $EL = (F, R_e, \alpha_e)$. Sei $|F| = m$. Seien $\underline{u}, \underline{v} \in P$ und $k \in \mathbb{N}_m$.

i) $\underline{v} \in P$ heißt zulässig genau dann, wenn gilt:

$$\underline{v} = \emptyset, \text{ oder}$$

$$\underline{v} = (v_1, \dots, v_k) \wedge \forall i, j \in \mathbb{N}_k: (i \neq j \Rightarrow v_i s_i \wedge v_j s_j = 0).$$

Ein zulässiger Knoten, der ein Blatt ist, heißt zulässiges Blatt.

Anders ausgedrückt: $\underline{v} = (v_1, \dots, v_k)$ ist ein zulässiges Blatt g.d.w.

$\underline{v}$ zulässig ist und $k = m$. Mit ZL wird die Menge aller zulässigen Knoten von VST bezeichnet.

$\underline{v} = (v_1, \dots, v_k) \in P$ heißt unzulässig g.d.w. gilt:

$$\exists i, j (i, j \in \mathbb{N}_k \wedge i \neq j \wedge (v_i s_i \wedge v_j s_j \neq 0)).$$

UZL bezeichnet die Menge aller unzulässigen Knoten von VST.

- ii) Eine Bahn $(\underline{u}^1, \dots, \underline{u}^l)$, $l \in \mathbb{N}_m$, heißt Zweig von VST genau dann, wenn gilt:
 $(\underline{u}^1 = \emptyset) \wedge (\underline{u}^1 \text{ ist ein zulässiger Knoten}).$

Ein Zweig ist vollständig, wenn $l = m$.

- iii) Sei $ZB \subseteq ZL$ die Menge aller zulässigen Blätter.

$$\hat{\omega} \equiv \text{Max} \{ \omega(\underline{v}) \mid \underline{v} \in ZB \} \text{ heißt Maximum.}$$

Jedes $\underline{v} \in ZB$ mit $\omega(\underline{v}) = \hat{\omega}$ heißt maximales Blatt.

Ein vollständiger Zweig heißt wesentlich, wenn sein Blatt maximal ist.

Mit MB wird die Menge aller maximalen Blätter von VST bezeichnet. D.h.

$$MB \hat{=} \{ \underline{v} \mid \underline{v} \in ZB \wedge \omega(\underline{v}) = \hat{\omega} \}.$$

- iv) Ein Teilbaum $T' = (P', A')$ heißt zulässig (unzulässig) genau dann, wenn seine Wurzel zulässig (unzulässig) ist.

Satz 2.4.15

Sei $VST = (P, A, F, \wp, \omega, \varphi, \emptyset)$ eine binäre Verzweigungsstruktur bezüglich $EL = (F, R_e, \alpha_e)$. Sei $|F| = m$ und $k \in \mathbb{N}_m$.

- i) $\emptyset$ und dessen beide Söhne sind zulässig.
- ii) $P = ZL \cup UZL$, $\emptyset = ZL \cap UZL$.
- iii) Sei $(\underline{u}, \underline{v}) \in A$. Sei $\underline{u}$ zulässig.
- $\underline{v} = \text{Nein-Sohn von } \underline{u} \implies \underline{v} \text{ ist zulässig.}$
- $\underline{v} = \text{Ja-Sohn von } \underline{u} \wedge (\wp(\underline{u}) \wedge \wp_1(\underline{v}) = 0) \implies \underline{v} \text{ ist zulässig.}$
- iv) Sei $(\underline{u}, \underline{v}) \in A$.
- $\underline{u} \text{ ist unzulässig} \implies \underline{v} \text{ ist unzulässig.}$
- $\underline{v} = \text{Ja-Sohn von } \underline{u} \wedge (\wp(\underline{u}) \wedge \wp_1(\underline{v}) \neq 0) \implies \underline{v} \text{ ist unzulässig.}$
- v) Sei $\underline{v}$ die Wurzel eines Teilbaumes $T' = (P', A')$.
- $\underline{v} \text{ ist unzulässig} \iff P' \subseteq UZL.$
- vi) Alle Knoten eines Zweiges sind zulässig.
- vii) Sei ZB die Menge der zulässigen Blätter von VST. Sei $\emptyset$ wie in Problem P2 angegeben.
- $ZB = \emptyset$.
- viii) Sei $\underline{y} \in \emptyset$. Sei $\underline{v} \in ZB$.
- $\underline{y} = \underline{v} \implies W(\underline{y}) = \omega(\underline{v}).$

Beweis:

- i) Dies ist eine unmittelbare Folgerung der Definition 2.4.14i.
 ii) Nach Definition 2.4.14i) ist $\underline{v} \in P$ entweder zulässig oder unzulässig.
 So wird P in zwei miteinander disjunkte Teilmengen, ZL und UZL, aufgeteilt.
 iii) Sei $\underline{u} = (u_1, \dots, u_k)$.

Teil 1: $\underline{v}$ ist der Nein-Sohn von $\underline{u}$

$$\Rightarrow \underline{v} = (v_1, \dots, v_k, 0), \text{ wobei } (v_1, \dots, v_k) = (u_1, \dots, u_k).$$

$$\Rightarrow \forall i, j \in \mathbb{N}_{k+1} (i \neq j \Rightarrow v_i s_i \wedge v_j s_j = 0) \quad (\underline{u} \text{ ist zulässig})$$

$$\Rightarrow \underline{v} \text{ ist zulässig.} \quad (\text{Def. 2.4.14i})$$

Teil 2: $\underline{v}$ ist der Ja-Sohn von $\underline{u}$ und $\varphi(\underline{u}) \wedge \varphi(\underline{v}) = 0$

$$\Rightarrow (\underline{v} = (v_1, \dots, v_k, 1)) \wedge (\bigvee_{i \in \mathbb{N}_k} v_i s_i \wedge s_{k+1} = 0) \quad (\text{Satz 2.4.13iii})$$

$$\text{wobei } (v_1, \dots, v_k) = (u_1, \dots, u_k).$$

$$\Rightarrow \bigvee_{i \in \mathbb{N}_k} (s_{k+1} \wedge v_i s_i) = 0$$

$$\Rightarrow \forall i, j \in \mathbb{N}_{k+1} (i \neq j \Rightarrow v_i s_i \wedge v_j s_j = 0) \quad (\underline{u} \text{ ist zulässig})$$

$$\Rightarrow \underline{v} \text{ ist zulässig.} \quad (\text{Def. 2.4.14i})$$

- iv) Sei $(\underline{u}, \underline{v}) \in A$. Sei $\underline{u} = (u_1, \dots, u_k)$.

Teil 1:

Annahme: $\underline{v} = (v_1, \dots, v_{k+1})$ ist zulässig.

$$\Rightarrow \forall i, j \in \mathbb{N}_{k+1} (i \neq j \Rightarrow v_i s_i \wedge v_j s_j = 0) \quad (\text{Def. 2.4.14i})$$

$$\Rightarrow \forall i, j \in \mathbb{N}_k (i \neq j \Rightarrow v_i s_i \wedge v_j s_j = 0)$$

$$\Rightarrow \forall i, j \in \mathbb{N}_k (i \neq j \Rightarrow u_i s_i \wedge u_j s_j = 0) \quad ((\underline{u}, \underline{v}) \in A)$$

$$\Rightarrow \underline{u} \text{ ist zulässig. Widerspruch.}$$

Teil 2: $\underline{v}$ ist der Ja-Sohn von $\underline{u}$ und $\varphi(\underline{u}) \wedge \varphi(\underline{v}) \neq 0$

$$\Rightarrow (\underline{v} = (u_1, \dots, u_k, 1)) \wedge (\bigvee_{i \in \mathbb{N}_k} u_i s_i \wedge s_{k+1} \neq 0) \quad (\text{Satz 2.4.13iii})$$

$$\Rightarrow \bigvee_{i \in \mathbb{N}_k} (v_i s_i \wedge v_{k+1} s_{k+1}) \neq 0$$

$$\Rightarrow \exists i \exists j (i, j \in \mathbb{N}_{k+1} \wedge i \neq j \wedge v_i s_i \wedge v_j s_j \neq 0)$$

$$\Rightarrow \underline{v} \text{ ist unzulässig.} \quad (\text{Def. 2.4.14i})$$

v) " $\Rightarrow$ "

$\underline{v}$ ist die Wurzel von T' und $\underline{v}$ ist unzulässig

$\Rightarrow$ Die Söhne von $\underline{v}$ sind unzulässig (Aussage iv)

$\Rightarrow$ Alle Nachfolger von $\underline{v}$ sind unzulässig

$\Rightarrow$ Alle Knoten von T' sind unzulässig

$\Rightarrow P' \subseteq \text{UZL.}$

" $\Leftarrow$ " Offensichtlich.

vi) Diese Aussage folgt aus Definition 2.4.14i und Aussage iv.

vii) Nach Def. 2.4.14i) enthält ZB alle Knoten $\underline{v} = (v_1, \dots, v_m) \in \{0, 1\}^m$, die zulässig sind, d.h.

$$\text{ZB} = \left\{ \underline{v} \mid \underline{v} \in \{0, 1\}^m \wedge \forall i, j \in \mathbb{N}_m (i \neq j \Rightarrow v_i \leq_i \wedge v_j \leq_j = \underline{0}) \right\}.$$

Diese Menge ist nicht anders als die Menge θ des Problems P2.

$$\text{viii) } W(\underline{y}) = \sum_{i \in I_{\underline{y}}} w_i \quad (\text{Bemerkung 2.4.5})$$

$$\omega(\underline{v}) = \sum_{i \in I_{\underline{v}}} w_i \quad (\text{Satz 2.4.13i})$$

Daraus folgt: $\underline{y} = \underline{v} \Rightarrow W(\underline{y}) = \omega(\underline{v})$. $\square$

Bemerkung 2.4.16

Die Aussagen iii, iv und v des Satzes 2.4.15 sind sehr nützliche Kriterien zum systematischen Test der Zulässigkeit der Knoten bzw. der Teilbäume. Bei dem noch anzugebenden "binären Verzweigungsprozeß" werden u.a. solche Kriterien benutzt, um alle unzulässigen Teilbäume von weiterer Betrachtung auszuschließen.

Satz 2.4.17

Sei $\text{VST} = (P, A, F, \mathcal{P}, \omega, \varphi, \underline{\theta})$ eine binäre Verzweigungsstruktur bezüglich $\text{EL} = (F, R_e, \alpha_e)$. Sei $\hat{\omega}$ das Maximum von VST. Sei MB die Menge der maximalen Blätter von VST. Sei $\hat{\theta} \times \{\hat{z}_q\}$ die Lösungsmenge des Problems P2.

$$\hat{\omega} = \hat{z}_q$$

$$\text{MB} = \hat{\theta}$$

Beweis:

Nach Aussage vii des Satzes 2.4.15 gilt $ZB = \theta$.

$$\begin{aligned}
 \hat{\omega} &= \text{Max} \{ \omega(\underline{v}) \mid \underline{v} \in ZB \} && (\text{Def. 2.4.14iii}) \\
 &= \text{Max} \{ W(\underline{y}) \mid \underline{y} \in \theta \} && (\text{Satz 2.4.15 viii}) \\
 &= \hat{Z}_q && (\text{Def. des Problems P2}) \\
 MB &= \{ \underline{v} \mid \underline{v} \in ZB \wedge \omega(\underline{v}) = \hat{\omega} \} && (\text{Def. 2.4.14iii}) \\
 &= \{ \underline{y} \mid \underline{y} \in \theta \wedge W(\underline{y}) = \hat{Z}_q \} && (\text{Satz 2.4.15viii}) \\
 &= \hat{\theta}. \quad \square && (\text{Def. des Problems P2})
 \end{aligned}$$

Bemerkung 2.4.18

Die Lösung des Problems P2 ist nach Satz 2.4.17 gleichbedeutend mit der Auffindung des Maximums und der Maximalen Blätter von VST. Zu diesem Zweck wird ein Rechenprozeß benötigt, der in der Lage sein soll, gezielte Suchoperationen auf die binäre Verzweigungsstruktur derart durchzuführen, daß MB und $\hat{\omega}$ mit wenig Rechenaufwand aufgefunden werden können. Bevor wir diesen Rechenprozeß definieren, geben wir zunächst ein Beispiel zur Veranschaulichung der bisher eingeführten Begriffe.

Bemerkung 2.4.19

Da $\hat{\omega} = \hat{Z}_q = \hat{Z}_p = \hat{Z}$, so werden wir künftig mit $\hat{Z}$ das Maximum der binären Verzweigungsstruktur bezeichnen.

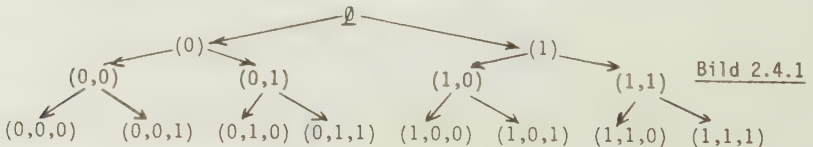
Beispiel 2.4.20

Sei $F = \{ ((1,0,0,1,1), 6), ((1,0,0,0,1), 2), ((0,1,0,1,0), 5) \}$.

Sei EL folgendermaßen angegeben:

$$\begin{aligned}
 L_1: & ((1,0,0,1,1), 6) \\
 & ((1,0,0,0,1), 2) \\
 L_2: & ((0,1,0,1,0), 5)
 \end{aligned}$$

Bild 2.4.1 zeigt den binären Verzweigungsbaum (P,A).



Die anderen Größen der binären Verzweigungsstruktur $VST = (P, A, F, \varphi, \omega, \psi, \bar{\varphi})$ sind in der folgenden Tabelle angegeben:

$\underline{v} \in P$	$\varphi_1(\underline{v})$	$\varphi_r(\underline{v})$	$\omega(\underline{v})$	$\varphi(\underline{v})$	$\bar{\varphi}(\underline{v})$	zulässig ?
$\emptyset$	(0,0,0,0,0)	0	0	(0,0,0,0,0)	11	Ja
(1)	(1,0,0,1,1)	6	6	(1,0,0,1,1)	5	Ja
(0)	(1,0,0,1,1)	6	0	(0,0,0,0,0)	7	Ja
(1,1)	(1,0,0,0,1)	2	8	(1,0,0,1,1)	5	Nein
(1,0)	(1,0,0,0,1)	2	6	(1,0,0,1,1)	5	Ja
(0,1)	(1,0,0,0,1)	2	2	(1,0,0,0,1)	5	Ja
(0,0)	(1,0,0,0,1)	2	0	(0,0,0,0,0)	5	Ja
(1,1,1)	(0,1,0,1,0)	5	13	(1,1,0,1,1)	0	Nein
(1,1,0)	(0,1,0,1,0)	5	8	(1,0,0,1,1)	0	Nein
(1,0,1)	(0,1,0,1,0)	5	11	(1,1,0,1,1)	0	Nein
(1,0,0)	(0,1,0,1,0)	5	6	(1,0,0,1,1)	0	Ja
(0,1,1)	(0,1,0,1,0)	5	7	(1,1,0,1,1)	0	Ja
(0,1,0)	(0,1,0,1,0)	5	2	(1,0,0,0,1)	0	Ja
(0,0,1)	(0,1,0,1,0)	5	5	(0,1,0,1,0)	0	Ja
(0,0,0)	(0,1,0,1,0)	5	0	(0,0,0,0,0)	0	Ja

Die vollständigen Zweige von VST sind:

$\underline{v} \in ZB$	$\omega(\underline{v})$
($\emptyset$, (1), (1,0), (1,0,0))	(1,0,0) 6
($\emptyset$, (0), (0,1), (0,1,1))	(0,1,1) 7
($\emptyset$, (0), (0,1), (0,1,0))	(0,1,0) 2
($\emptyset$, (0), (0,0), (0,0,1))	(0,0,1) 5
($\emptyset$, (0), (0,0), (0,0,0))	(0,0,0) 0

Der wesentliche Zweig ist ($\emptyset$, (0), (0,1), (0,1,1)).

Das maximale Blatt ist (0,1,1).

Das Maximum ist 7.

Für den Rest dieses Abschnitts sei $VST = (P, A, F, \wp, \varphi, \xi)$ eine binäre Verzweigungsstruktur bezüglich $EL = (F, R_e, \sim_e)$. Sei $|F| = m$. Sei $\hat{Z}$ das Maximum und MB die Menge der maximalen Blätter von VST. Sei ZL bzw. UZL die Menge der zulässigen Knoten bzw. unzulässigen Knoten von VST.

Definition 2.4.21

Sei $T = (P, A)$ ein binärer Verzweigungsbaum. Eine Markierungsregel auf T ist ein Tripel $MR = (KM, \xi, \delta)$ bestehend aus

einer Menge von Knotenmarken $KM \cong \{L, R, B, \lambda\}$, wobei

$L \cong$ linksvorwärts

$R \cong$ rechtsvorwärts

$B \cong$ rückwärts

$\lambda \cong$ leere Marke ,

einer Knotenmarkierungsfunktion

$\xi: P \times KM \rightarrow KM$, und

einer Knotenauswahlfunktion

$\delta: P \times KM \rightarrow P$.

$PM \subseteq P \times KM$ wird genau dann markierte Knotenmenge genannt, wenn

$$PM = \{(x, y) \mid x \in P \wedge y \in \{L, R, B\}\}.$$

$PN \subseteq P \times KM$ wird genau dann nicht markierte Knotenmenge genannt, wenn

$$PN = \{(x, y) \mid x \in P \wedge y = \lambda\}.$$

Definition 2.4.22

Eine Suchstrategie auf VST ist ein Paar $SU = (MR, AS)$. Dabei ist $MR = (KM, \xi, \delta)$ eine Markierungsregel auf $T = (P, A)$ und $AS \in \mathbb{N}_0$ eine Anfangsschranke, für die gilt:

$$0 \leq AS < \hat{Z}.$$

Bemerkung 2.4.23

AS läßt sich mittels eines heuristischen Verfahrens wie z.B. /118/ auffinden. Falls kein heuristisches Programm dazu vorhanden ist, kann man stets $AS = 0$ annehmen.

Definition 2.4.24

Ein binärer Verzweigungsprozeß ist ein Tripel $VP = (VST, SU, \mathcal{E})$, wobei VST eine binäre Verzweigungsstruktur und SU eine Suchstrategie auf VST ist.

$\mathcal{E}: N_0 \rightarrow [0,1)$ heißt Fehlerfunktion von VP. $\mathcal{E}(US) \in [0,1)$ heißt relativer Fehler bezüglich US. US heißt untere Schranke von VP, für die gilt:

$$US = \begin{cases} \text{Max} \{ \omega(\underline{v}) \mid \underline{v} \in \text{MBL} \}, & \text{wenn } \text{MBL} \neq \emptyset. \\ \text{AS}, & \text{wenn } \text{MBL} = \emptyset. \end{cases}$$

Dabei ist MBL die Menge aller markierten zulässigen Blätter von VST.

VP heißt exakt (approximierend), wenn $\mathcal{E}(US) = 0$ ($\mathcal{E}(US) > 0$).

Bemerkung 2.4.25

In diesem Kapitel werden wir uns ausschließlich auf die Eigenschaften des exakten binären Verzweigungsprozesses konzentrieren. Dieser VP liefert entweder die exakte(n) Lösung(en) des Problems P2 oder die Aussage, daß für P2 keine Lösung existiert. Im nächsten Kapitel (Kap. 3) werden wir ausführlich approximierende binäre Verzweigungsprozesse bzw. Verzweigungsprozesse, die andere Bedingungen zu erfüllen haben, untersuchen.

Eine Suchstrategie wird im wesentlichen von ihrer Markierungsregel bestimmt. Die Markierungsregel wird ihrerseits durch die Knotenmarkierungsfunktion und die Knotenauswahlfunktion festgelegt. Eine Markierungsregel, bei der ξ und δ genau angegeben sind, bekommt eine eindeutige Bezeichnung.

Definition 2.4.26

Eine Markierungsregel (KM, ξ, δ) wird g.d. MAZU (Markieren und Zurückkehren) genannt, wenn $\xi: P \times KM \rightarrow KM$ und $\delta: P \times KM \rightarrow P$ folgendermaßen angegeben werden:

Fall 1: $\underline{u} = \emptyset \wedge \exists v((\underline{u}, v) \in A)$

$$(\underline{u}, \lambda) \in P \times KM \Rightarrow \delta(\underline{u}, \lambda) = \underline{v} \wedge \xi(\underline{u}, \lambda) = R \wedge \underline{v} = \text{Ja-Sohn von } \underline{u} \quad (1)$$

$$(\underline{u}, R) \in P \times KM \Rightarrow \delta(\underline{u}, R) = \underline{v} \wedge \xi(\underline{u}, R) = L \wedge \underline{v} = \text{Nein-Sohn von } \underline{u} \quad (2)$$

$$(\underline{u}, L) \in P \times KM \Rightarrow \delta(\underline{u}, L) = \underline{u} \wedge \xi(\underline{u}, L) = B \quad (3)$$

$$(\underline{u}, B) \in P \times KM \Rightarrow \delta(\underline{u}, B) = \underline{u} \wedge \xi(\underline{u}, B) = B \quad (4)$$

Fall 2: $u \neq \emptyset \wedge \exists v \exists t ((u, v) \in A \wedge (t, u) \in A)$

Fall 2.1: $u \in ZL \wedge \omega(u) + \xi(u) > US$

$$(u, \lambda) \in P \times KM \Rightarrow \delta(u, \lambda) = \underline{v} \wedge \xi(u, \lambda) = R \wedge \underline{v} = \text{Ja-Sohn von } u \quad (5)$$

$$(u, R) \in P \times KM \Rightarrow \delta(u, R) = \underline{v} \wedge \xi(u, R) = L \wedge \underline{v} = \text{Nein-Sohn von } u \quad (6)$$

$$(u, L) \in P \times KM \Rightarrow \delta(u, L) = \underline{t} \wedge \xi(u, L) = B \quad (7)$$

Für $(u, B) \in P \times KM$ sind δ und ξ undefiniert.

Fall 2.2: $(u \in ZL \wedge \omega(u) + \xi(u) \leq US) \vee (u \in UZL)$

$$(u, \lambda) \in P \times KM \Rightarrow \delta(u, \lambda) = \underline{t} \wedge \xi(u, \lambda) = B \quad (8)$$

Für (u, R) , (u, L) und (u, B) sind δ und ξ undefiniert.

Fall 3: u ist ein Blatt $\wedge (t, u) \in A$

$$(u, \lambda) \in P \times KM \Rightarrow \delta(u, \lambda) = \underline{t} \wedge \xi(u, \lambda) = B \quad (9)$$

Für (u, R) , (u, L) und (u, B) sind δ und ξ undefiniert.

(1), (2), --, (9) werden als Vorschriften der Suchstrategie (MAZU, AS) auf VST bezeichnet.

Definition 2.4.27

Sei $VP = (VST, SU, 0)$ ein binärer Verzweigungsprozeß. Sei $SU = (MAZU, AS)$.

Seien $a, b \in \{L, R, B, \lambda\}$. Sei $\delta(u, a) = \underline{v}$ und $\xi(u, a) = b$.

VP ist im Zustand laufbereit, wenn $u = \underline{v} = \emptyset$ und $a = b = B$. In allen anderen Fällen, in denen δ und ξ durch die Vorschriften (1), (2), (5), (6), (7), (8), (9) festgelegt werden, ist VP im Zustand laufend. Ein VP, der sich im Zustand laufend (laufbereit) befindet, heißt laufender (laufbereiter) Verzweigungsprozeß.

Jeder Knoten $u \in P$ besitzt 3 Zustände: aktiv, inaktiv und wartend. u ist g.d. (im Zustand) aktiv, wenn $(u, L) \in PM$ oder $(u, R) \in PM$; d.h. wenn u entweder mit L oder mit R markiert ist. u ist genau dann (im Zustand) inaktiv, wenn $(u, B) \in PM$. u ist genau dann (im Zustand) wartend, wenn $(u, \lambda) \in PN$. Wartend ist der Anfangszustand aller Knoten der binären Verzweigungsstruktur.

Ein Übergang von Knoten u zu Knoten $v \neq u$ ist genau dann möglich, wenn $\delta(u, a) = v$ und $\xi(u, a) = b$. u heißt Ausgangsknoten und v heißt Zielknoten. Ein VP markiert den Knoten u , wenn u der Ausgangsknoten von VP ist. Jeder Übergang wird als ein Schritt des VP bezeichnet.

Ein laufender VP geht einen Gang durch die VST, wenn er von der Wurzel $\emptyset$ ausgehend gemäß der Suchstrategie SU von einem Knoten zu einem anderen übergeht und bei jedem Schritt den Ausgangsknoten markiert. Ein Gang ist genau dann eine Tour, wenn VP wieder zur Wurzel zurückkehrt und sich dort in den Zustand laufbereit versetzt. Eine Folge von Knoten $(\underline{u}^1, \dots, \underline{u}^k)$ heißt genau dann Spur des VP, wenn $\underline{u}^1 = \underline{u}^k = \emptyset$ und $\forall i \in \mathbb{N}_{k-1} (\underline{u}^i = \text{Ausgangsknoten} \Rightarrow \underline{u}^{i+1} = \text{Zielknoten})$. Das Zeitintervall, in dem eine Tour des VP stattfindet, heißt Laufphase. Die Anzahl der Schritte, die der VP bei einer Tour machen muß, heißt Aufwand des VP. Der Aufwand wird mit $AW(VP)$ bezeichnet.

Bemerkung 2.4.28

Die Knotenmarkierungen sind notwendige "Gedächtnishilfen" für den laufenden VP, um eine Tour mit möglichst wenig Aufwand zu machen. Man kann sich das so vorstellen, daß vor dem Beginn einer Tour alle Knoten von VST den "Besuch" des VP erwarten (alle Knoten befinden sich im Zustand wartend). Der VP fängt seinen Gang immer von der Wurzel der VST an und versucht, eine Tour mit dem kleinsten Aufwand zu machen. Von einem gegebenen Knoten aus kann der VP gemäß den in SU festgelegten Vorschriften zu einem der 3 möglichen Zielknoten übergehen: zum Ja-Sohn (rechtsvorwärts), zum Nein-Sohn (linksvorwärts) oder zum Vater (rückwärts); dabei wird der Ausgangsknoten mit der "Richtung" des jeweiligen Übergangs markiert. Damit wird garantiert, daß der gleiche Übergang nicht wiederholt wird, wenn der VP eventuell irgendwann später zu diesem Knoten zurückkehrt.

Satz 2.4.29

Sei $VP = (VST, SU, 0)$ ein binärer Verzweigungsprozeß. Sei $\underline{u} \in P$.

- i) $\underline{u}$ ist wartend $\Rightarrow$ Alle Nachfolger von $\underline{u}$ sind wartend.
- ii) Sei $\underline{u} \neq \emptyset$.
 $\underline{u}$ ist inaktiv $\Rightarrow$ Es gibt keinen Übergang zu $\underline{u}$.
- iii) $\underline{u}$ ist unzulässig $\Rightarrow$ Alle echten Nachfolger von $\underline{u}$ sind wartend.
- iv) $\underline{u}$ ist inaktiv und $\omega(\underline{u}) + \varphi(\underline{u}) \leq US$
 $\Rightarrow$ Alle echten Nachfolger von $\underline{u}$ sind wartend.
- v) Ein markiertes Blatt ist inaktiv.
- vi) Bei einer Tour wird jedes Blatt höchstens einmal markiert.

- vii) Bei einer Tour wird jeder Knoten der VST höchstens dreimal markiert.
- viii) Nach einer Tour befinden sich alle Knoten der VST entweder im Zustand inaktiv oder im Zustand wartend.
- ix) Ein laufender VP wiederholt nicht denselben Schritt bei einer Tour.
- x) Die Gesamtzahl der Schritte, die der VP bei einer Tour machen muß, ist endlich.

Beweis:

- i) Nach Def. 2.4.27 ist "wartend" der Anfangszustand aller Knoten der VST. Bei einem Übergang von Vater zu Sohn muß der Vater markiert werden (d.h. Vater ist nicht mehr wartend). Solange $\underline{u}$ nicht markiert ist, können die echten Nachfolger von $\underline{u}$ niemals markiert werden.
- ii) Angenommen, es gibt einen Übergang zu $\underline{u}$, der inaktiv ist. Nach Def. 2.4.26 kann dies nur geschehen, wenn $\underline{u} = \emptyset$. Dies steht im Widerspruch zu der Voraussetzung, daß $\underline{u} \neq \emptyset$.
- iii) Nach Def. 2.4.26, Fall 2.2, markiert VP einen wartenden, unzulässigen Knoten mit B (d.h. der Knoten wird inaktiv) und geht zu dessen Vater über. Es kann danach keinen Übergang zu $\underline{u}$ mehr geben (Aussage ii). Die echten Nachfolger von $\underline{u}$ können deshalb nicht bei derselben Tour markiert werden.
- iv) Wie iii). In diesem Falle ist $\underline{u}$ zulässig und die Bedingung $\omega(\underline{u}) + \bar{\omega}(\underline{u}) \leq US$ wird erfüllt (Fall 2.2 der Definition 2.4.26).
- v) Ist $\underline{u}$ ein Blatt, so erhält $\underline{u}$ als Ausgangsknoten des VP die Marke B und VP geht zum Vater von $\underline{u}$ über (Fall 3 der Definition 2.4.26). Dies bedeutet, daß ein markiertes Blatt inaktiv ist.
- vi) Bei einer Tour gibt es für ein Blatt zwei Möglichkeiten: Das Blatt wird nicht markiert oder es wird markiert und bleibt danach inaktiv (Aussage v). Im letzten Falle kann es keinen Übergang zu diesem Blatt mehr geben (Aussage ii).
- vii) Ist $\underline{u}$ ein Blatt, so wird $\underline{u}$ nach Aussage vi höchstens einmal markiert. Ist $\underline{u}$ kein Blatt und $\underline{u} \neq \emptyset$, so kann $\underline{u}$ nach Def. 2.4.26 nacheinander die Marke R, L und B erhalten. Mit B wird $\underline{u}$ inaktiv, danach kann $\underline{u}$ nicht mehr markiert werden (Aussage ii).
Gilt $\underline{u} = \emptyset$, so wird $\underline{u}$ nach Def. 2.4.26 (Fall 1) genau dreimal markiert.

viii) Annahme: Nach einer Tour gibt es einen Knoten $\underline{u} \in P$, der weder inaktiv noch wartend ist.

Daraus folgt: $\underline{u}$ ist aktiv (Def. 2.4.27)

$$\Rightarrow (\underline{u}, R) \in PM \vee (\underline{u}, L) \in PM$$

$$\Rightarrow VP \text{ ist laufend. Widerspruch.}$$

(Nach einer Tour befindet sich VP im Zustand laufbereit).

ix) Ein Schritt ist nach Def. 2.4.27 ein Übergang von $\underline{u}$ zu $\underline{v} \neq \underline{u}$, bei dem gilt: $\delta(\underline{u}, a) = \underline{v}$ und $\xi(\underline{u}, a) = b$. Für einen laufenden VP gilt außerdem: $a \neq b$. D. h. die Markierung von $\underline{u}$ wird bei jedem Übergang, bei dem $\underline{u}$ der Ausgangsknoten ist, geändert, sodaß $\underline{v}$ bestimmt nicht der Zielknoten des nächsten Übergangs sein wird, der von $\underline{u}$ ausgeht.

x) Diese Aussage ergibt sich aus den Aussagen vi, vii, ix und der Def. 2.4.7 (die Knotenmenge des binären Verzweigungsbaums ist endlich). $\square$

Bemerkung 2.4.30

Es wird angenommen, daß jeder Schritt des VP eine Zeiteinheit benötigt. Die Aussage x) des Satzes 2.4.29 ist gleichbedeutend mit der Aussage, daß die Laufphase eines VP endlich ist.

Satz 2.4.31

Sei IA die Menge der Knoten, die nach einer Tour des VP inaktiv sind. Sei $AW(VP)$ der Aufwand des VP.

$$AW(VP) \leq 3 |IA|$$

Beweis:

Nach Aussage viii des Satzes 2.4.29 befinden sich alle Knoten der VST nach einer Tour entweder im Zustand inaktiv oder im Zustand wartend. Bei einer Tour werden ausschließlich diejenigen Knoten markiert, die in IA enthalten sind (Aussagen ii, iii und iv). Jeder Knoten wird nach Aussage vii des Satzes 2.4.29 höchstens dreimal markiert. Für jede Markierung benötigt der VP einen Schritt. Dies bedeutet, daß die Gesamtzahl der Schritte (d.h. der Aufwand) $\leq 3 |IA|$ sein muß. $\square$

Bemerkung 2.4.32

Nach Satz 2.4.31 läßt sich die Anzahl der inaktiven Knoten nach einer Tour als Aufwand des VP betrachten. Diese Betrachtungsweise bringt den Vorteil mit sich, daß die Aufzählung der einzelnen Schritte des VP bei einer Tour nicht benötigt wird. Um $|IA|$ zu erhalten, braucht man nur die Anzahl der inaktiv gewordenen Knoten zu registrieren. Die nach einer Tour noch im Zustand "wartend" befindlichen Knoten gehören zu Teilbäumen, die vom binären Verzweigungsprozeß ausgeschlossen sind; sie werden als nicht existent betrachtet. Da man sich in den meisten Fällen ausschließlich für eine Abschätzung des Aufwands bzw. für dessen oberen Grenzwert interessiert, wird $|IA|$ bei den darauffolgenden Diskussionen als Aufwand betrachtet.

Für den folgenden Satz nehmen wir ohne Einschränkung der Allgemeinheit an, daß der VP den Gang stets zum Zeitpunkt 0 beginnt.

Satz 2.4.33

Sei $[0, t]$ mit $t \in \mathbb{R}^+$ eine Laufphase des binären Verzweigungsprozesses $VP = (VST, SU, 0)$. Seien $i, j \in \mathbb{N}$ und $t_i, t_j \in [0, t]$. Seien $US(t_i)$, $US(t_j)$ die unteren Schranken zum Zeitpunkt t_i bzw. t_j .

$$i) \quad t_i \geq t_j \Rightarrow US(t_i) \geq US(t_j)$$

$$ii) \quad US(t) = \hat{Z}$$

Beweis:

i) Sei $MBL(t_i)$ ($MBL(t_j)$) die Menge der markierten zulässigen Blätter von VST zum Zeitpunkt t_i (t_j).

Aus Def. 2.4.26 folgt $t_i \geq t_j \Rightarrow MBL(t_j) \subseteq MBL(t_i)$.

Aus Def. 2.4.24 folgt $MBL(t_j) \subseteq MBL(t_i) \Rightarrow US(t_i) \geq US(t_j)$.

ii) Dies ist eine unmittelbare Folgerung der Aussage i). $\square$

Satz 2.4.34

Sei $VP = (VST, SU, 0)$. Sei $SU = (MAZU, AS)$. Sei RK_i , $i \in \mathbb{N}_k$, der Rang der F-Liste L_j in EL, wobei $\alpha_1(L_j) = i$. Sei τ der Zeitpunkt, zu dem die erste untere Schranke $US(\tau) \leq AS$ aufgefunden wird.

$$US(\tau) \leq \hat{Z} \leq \sum_{i \in \mathbb{N}_k} RK_i$$

Beweis: Offensichtlich. $\square$

Bemerkung 2.4.35

$US(\tau)$ bildet den unteren Grenzwert und $\sum_{i \in IN_k} RK_i$ den oberen Grenzwert von $\hat{Z}$.

Die spezielle Struktur der E-Liste EL, auf die sich VST bezieht, ist ausschlaggebend für die Bestimmung der Markierungsregel der Suchstrategie SU. EL und SU sind so aufeinander abgestimmt, daß die folgenden Forderungen erfüllt werden können:

- i) Das Problem P2 soll mit dem kleinsten Aufwand gelöst werden.
- ii) Eine genügend große untere Schranke soll bei einer Tour möglichst früh aufgefunden werden. Wenn die Zeit für eine Tour die zulässige Rechenzeit übersteigt, sodaß die Tour frühzeitig abgebrochen werden muß, soll die bis zum Zeitpunkt des Abbruchs erhaltene untere Schranke und das entsprechende Blatt als eine brauchbare approximierende Lösung des Problems P2 betrachtet werden können.

Bei den in der Praxis auftretenden Problemen kommt es öfter vor, daß eine untere Schranke, die dem Maximum der VST entspricht, bereits kurz nach Beginn der Tour aufgefunden wird. Der Gang des VP muß in diesem Falle trotzdem fortgesetzt werden, da der Nachweis, daß eine untere Schranke tatsächlich dem Maximum entspricht, erst erbracht werden kann, nachdem der VP seine Tour beendet hat (Satz 2.4.33).

An dieser Stelle scheint es angebracht, die Ideen, auf denen der binäre Verzweigungsprozeß und die damit zusammenhängenden Begriffe und Eigenschaften beruhen, mittels einer Analogie zu veranschaulichen:

Ein wilder Affe (der binäre Verzweigungsprozeß VP) hat die Absicht, die leckeren Früchte (die Blockindikatoren der E-Liste) eines Obstbaums (des binären Verzweigungsbaums) zu stehlen. Dabei versucht er, bei seinem "Vorhaben" eine Menge von besten reifen Früchten des Obstbaums (die Lösungsmenge des Problems P2) mit möglichst wenig Anstrengung (Aufwand des VP) zu fressen bzw. zu sammeln. Wie jeder schlaue Dieb, der sich vor einem riskanten Unternehmen ein klares Bild seiner künftigen Beute verschafft, weiß der Affe nach einer langen Beobachtungszeit über alle mit der Struktur des Obstbaums zusammenhängenden Merkmale (die binäre Verzweigungsstruktur) ziemlich genau Bescheid. Er besitzt z.B. ausreichende Kenntnisse über die Qualität (angegeben durch die Abbildungen ω und φ) der Früchte und darüber, wie sie auf dem Baum verteilt sind (angegeben durch die Abbildungen $\mathcal{F}$ und $\mathcal{X}$). Die Früchte des Obstbaums sind

nämlich nur an dessen Knoten bzw. an dessen Blättern zu finden. Solche Kenntnisse ermöglichen dem Affen u.a., reife und große Früchte (zulässige Knoten mit großem ω) von unreifen, kleinen oder faulen Früchten (unzulässige Knoten bzw. Knoten mit kleinem ω) zu unterscheiden, sodaß er sich bei jedem Knoten, den er bei seiner Klettertour (Tour des VP) erreicht hat, entscheiden kann, ob die dort vorhandenen Früchte gefressen bzw. mitgenommen werden sollen.

Die Klettertour des Affen ist leider mit ziemlich großem Risiko verbunden. Der Bauer (ein Prozeß im Rechensystem, der die Rechenzeit und den Speicherbedarf des Verzweigungsprozesses überwacht), dem der Obstbaum gehört, kontrolliert den Baum regelmäßig. Er wird jeden Dieb, den er beim Stehlen seiner Früchte erwischt, sofort erschießen (der VP wird inaktiviert und aus dem Arbeitsspeicher verdrängt). Da dem Affen diese Gefahr durchaus bewußt ist, sieht er sich gezwungen, einen genauen Plan (die Suchstrategie) für sein Vorhaben "auszuarbeiten". Der Plan enthält im wesentlichen eine Strategie für die Klettertour, die darauf abzielt, eine Menge von besten reifen Früchten (die Lösungsmenge) möglichst schnell und ohne allzu große Anstrengung (möglichst wenig Aufwand) zu stehlen. Dabei wird berücksichtigt, daß er noch Zeit genug hat, mit einer Menge von Früchten zu fliehen, wenn der Bauer bei seiner Klettertour plötzlich auftaucht (vorzeitiger Abbruch der Tour, wobei sich eine approximierende Lösung ergibt). Der Plan legt u.a. fest, wie sich der Affe von Knoten zu Knoten bewegt (die Knotenauswahlfunktion δ). Als Gedächtnishilfe markiert der Affe jeden Knoten (die Knotenmarkierungsfunktion ξ), von dem er ausgeht, sodaß er sich von einem Knoten aus nicht zum zweiten Mal in eine gegebene Richtung bewegt. Darüberhinaus versucht der Affe, unnötige Anstrengungen bei der Klettertour zu vermeiden, indem er diejenigen Teilbäume von seiner Klettertour ausschließt, bei denen mit Sicherheit keine großen reifen Früchte zu finden sind (z.B. Teilbäume mit unzulässiger Wurzel).

Beispiel 2.4.36

Wir betrachten die im Beispiel 2.3.45 angegebene E-Liste.

Sei $VST = (P, A, F, \wp, \omega, \varphi, \mathcal{E})$ eine binäre Verzweigungsstruktur bezüglich dieser E-Liste. Sei $SU = (MAZU, 0)$ eine Suchstrategie auf VST.

Sei $VP = (VST, SU, 0)$ der binäre Verzweigungsprozeß.

In Bild 2.4.2 wird der binäre Verzweigungsbaum $T = (P, A)$ dargestellt, bei dem ausschließlich diejenigen Knoten gezeigt werden, die sich nach einer Tour des VP im Zustand "inaktiv" befinden. Mit Ausnahme von $\emptyset$ werden die Knoten entweder als Ja-Knoten oder als Nein-Knoten (Def. 2.4.38) bezeichnet. Die mit x gekennzeichneten Knoten sind unzulässig, d.h. sie sind die Wurzeln der unzulässigen Teilbäume. Alle anderen Knoten der unzulässigen Teilbäume erscheinen nicht in Bild 2.4.2. Das Zeichen $\checkmark$ kennzeichnet einen zulässigen Knoten $\underline{v}$, bei dem die Bedingung $\omega(\underline{v}) + \hat{\varphi}(\underline{v}) \leq US$ erfüllt ist. Ist $\underline{v}$ ein maximales Blatt, so werden $I_{\underline{v}}$ und $\hat{Z}$ angegeben. Die Blockindikatoren der E-Liste werden durch ihre Indizes $i \in \mathbb{N}_{22}$ angegeben. i kennzeichnet zugleich die Höhe der entsprechenden Knoten von T .

Das Ergebnis:

Das maximale Blatt: $\underline{v} = (1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0)$.

Das Maximum: $\hat{Z} = 165$.

Der Aufwand : 352 Knoten, davon ein einziges zulässiges Blatt, das maximal ist.

(Zum Vergleich: $|P| = 2^{23} - 1$ Knoten !!)

Das maximale Blatt entspricht den folgenden Blöcken des in Beispiel 2.3.45 angegebenen Netzes:

$$\{2, 3\} ; \{5, 6\} ; \{8, 9, 10\}.$$

Bemerkung 2.4.37

- i) $\underline{v}$ wird bereits bei einem Aufwand von 42 Knoten markiert. Dies entspricht ca. 12% des Gesamtaufwands. Der Rest des Aufwands wird dazu benötigt, $\underline{v}$ als maximal nachzuweisen.
- ii) Der Aufwand des VP für eine Tour wird im wesentlichen von der Höhe des binären Verzweigungsbaums bestimmt, welche die Anzahl der Blockindikatoren in der E-Liste angibt. Es stellt sich nun die Frage, ob alle diese Blockindikatoren notwendig sind, um das Maximum festzulegen. Im nächsten Abschnitt (Abs. 2.5) wird diese Frage eingehend untersucht.
- iii) Ein anderer Weg zur Reduzierung des Aufwands besteht darin, eine bessere Suchstrategie zu entwickeln. Zu diesem Zweck kann man eine neue bessere Markierungsregel festlegen oder eine größere Anfangsschranke wählen. Für den Rest dieses Abschnitts werden wir diese beiden Möglichkeiten untersuchen.

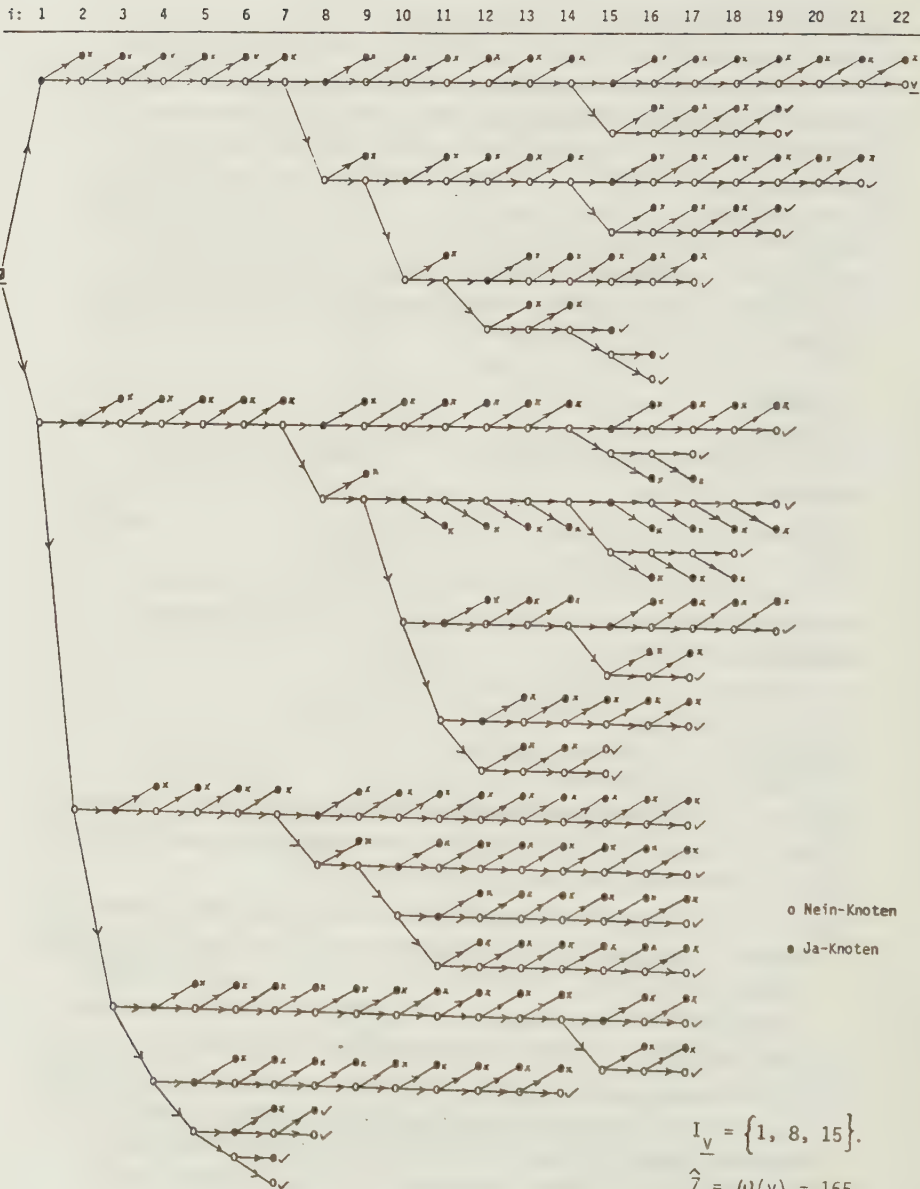


Bild 2.4.2

Definition 2.4.38

$\underline{u} \in P$ wird genau dann Ja-Knoten (Nein-Knoten) genannt, wenn $\underline{u}$ einen Vater besitzt, dessen Ja-Sohn (Nein-Sohn) $\underline{u}$ ist.

Folgerung 2.4.39

Alle Knoten $\underline{u} \in P \setminus \{\emptyset\}$ sind entweder Ja-Knoten oder Nein-Knoten.

Bemerkung 2.4.40

$\emptyset$ ist weder Ja-Knoten noch Nein-Knoten. In Bild 2.4.2 des Beispiels 2.4.36 sind alle Ja-Knoten mit "•", alle Nein-Knoten mit "o" gekennzeichnet.

Definition 2.4.41

Eine Markierungsregel (KM, ξ, δ) wird genau dann MARZ (Markieren, Reduzieren und Zurückkehren) genannt, wenn $\xi: P \times KM \rightarrow KM$ und $\delta: P \times KM \rightarrow P$ folgendermaßen angegeben werden:

Fall 1: $\underline{u} = \emptyset \wedge \exists v((\underline{u}, v) \in A)$

$$(\underline{u}, \lambda) \in P \times KM \Rightarrow \delta(\underline{u}, \lambda) = \underline{v} \wedge \xi(\underline{u}, \lambda) = R \wedge \underline{v} = \text{Ja-Sohn von } \underline{u} \quad (1')$$

$$(\underline{u}, R) \in P \times KM \Rightarrow \delta(\underline{u}, R) = \underline{v} \wedge \xi(\underline{u}, R) = L \wedge \underline{v} = \text{Nein-Sohn von } \underline{u} \quad (2')$$

$$(\underline{u}, L) \in P \times KM \Rightarrow \delta(\underline{u}, L) = \underline{u} \wedge \xi(\underline{u}, L) = B \quad (3')$$

$$(\underline{u}, B) \in P \times KM \Rightarrow \delta(\underline{u}, B) = \underline{u} \wedge \xi(\underline{u}, B) = B \quad (4')$$

Fall 2: $\underline{u} \neq \emptyset \wedge \exists F_j \exists v \exists t(\vartheta(\underline{u}) \in F_j \wedge (\underline{u}, v) \in A \wedge (t, \underline{u}) \in A)$

Fall 2.1: $\underline{u} \in ZL \wedge \omega(\underline{u}) + \bar{x}(\underline{u}) > US \wedge \vartheta(\underline{v}) \in F_j \wedge ((\underline{u} = \text{Ja-Knoten}) \vee (\underline{u} = \text{Nein-Knoten} \wedge \exists \underline{s}(\underline{s} \in ZL \wedge \vartheta(\underline{s}) \in F_j \wedge \underline{s} = \text{Vorgänger von } \underline{u} \wedge \underline{s} = \text{Ja-Knoten})))$.

$$(\underline{u}, \lambda) \in P \times KM \Rightarrow \delta(\underline{u}, \lambda) = \underline{v} \wedge \xi(\underline{u}, \lambda) = L \wedge \underline{v} = \text{Nein-Sohn von } \underline{u} \quad (5')$$

$$(\underline{u}, L) \in P \times KM \Rightarrow \delta(\underline{u}, L) = t \wedge \xi(\underline{u}, L) = B \quad (6')$$

Für $(\underline{u}, R)$ und $(\underline{u}, B)$ sind δ und ξ undefiniert.

Fall 2.2: $\underline{u} \in ZL \wedge \omega(\underline{u}) + \bar{x}(\underline{u}) > US \wedge ((\vartheta(\underline{v}) \notin F_j) \vee (\vartheta(\underline{v}) \in F_j \wedge \underline{u} = \text{Nein-Knoten} \wedge \forall \underline{s}(\vartheta(\underline{s}) \in F_j \wedge \underline{s} = \text{Vorgänger von } \underline{u} \wedge \underline{s} = \text{Nein-Knoten})))$.

$$(\underline{u}, \lambda) \in P \times KM \Rightarrow \delta(\underline{u}, \lambda) = \underline{v} \wedge \xi(\underline{u}, \lambda) = R \wedge \underline{v} = \text{Ja-Sohn von } \underline{u} \quad (7')$$

$$(\underline{u}, R) \in P \times KM \Rightarrow \delta(\underline{u}, R) = \underline{v} \wedge \xi(\underline{u}, R) = L \wedge \underline{v} = \text{Nein-Sohn von } \underline{u} \quad (8')$$

$$(\underline{u}, L) \in P \times KM \Rightarrow \delta(\underline{u}, L) = t \wedge \xi(\underline{u}, L) = B \quad (9')$$

Für $(\underline{u}, B)$ sind δ und ξ undefiniert.

Fall 2.3: $(\underline{u} \in ZL \wedge \omega(\underline{u}) + \Phi(\underline{u}) \leq US) \vee (\underline{u} \in UZL).$

$$(\underline{u}, \lambda) \in P \times KM \implies \delta(\underline{u}, \lambda) = \underline{t} \wedge \xi(\underline{u}, \lambda) = B \quad (10')$$

Für $(\underline{u}, R)$, $(\underline{u}, L)$ und $(\underline{u}, B)$ sind δ und ξ undefiniert.

Fall 3: $\underline{u}$ ist ein Blatt $\wedge (\underline{t}, \underline{u}) \in A.$

$$(\underline{u}, \lambda) \in P \times KM \implies \delta(\underline{u}, \lambda) = \underline{t} \wedge \xi(\underline{u}, \lambda) = B \quad (11')$$

Für $(\underline{u}, R)$, $(\underline{u}, L)$ und $(\underline{u}, B)$ sind δ und ξ undefiniert.

Bemerkung 2.4.42

MARZ unterscheidet sich von MAZU nur im Falle 2, in dem $\underline{u}$ sowohl einen Vater als auch einen Sohn besitzt. Der Fall 2.2 von MAZU ist identisch mit dem Fall 2.3 von MARZ. Die Bedingungen der beiden Fälle 2.1 und 2.2 von MARZ werden aus dem Fall 2.1 von MAZU abgeleitet, indem zusätzliche Tests für den Vorgänger und für den Sohn von $\underline{u}$ durchgeführt werden. Sind alle Bedingungen des Falls 2.1 von MARZ erfüllt, dann werden statt 7', 8', 9' (identisch mit 5, 6, 7) die Vorschriften 5' und 6' für den Gang verwendet. In allen anderen Fällen verhält sich der VP bei seinem Gang so, als wäre seine Suchstrategie von der Markierungsregel MAZU allein bestimmt.

Bemerkung 2.4.43

Sei $SU' = (MARZ, AS)$. Sei $VP' = (VST, SU', 0)$. Die Aussagen der Sätze 2.4.29, 2.4.31 und 2.4.33 gelten offensichtlich auch für VP' .

Lemma 2.4.44

Sei $(\underline{u}, \underline{v}) \in A$. Sei $\underline{u} \in ZL \setminus \{\emptyset\}$. Seien $\mathcal{P}(\underline{v}), \mathcal{P}(\underline{u}) \in F_j$.

$\underline{u} = \text{Ja-Knoten} \wedge \underline{v} = \text{Ja-Sohn von } \underline{u} \implies \underline{v} \in UZL.$

Beweis:

$\mathcal{P}(\underline{u}), \mathcal{P}(\underline{v}) \in F_j$ bedeutet, daß die j -te Komponente von $\mathcal{P}_1(\underline{u})$ bzw. von $\mathcal{P}_1(\underline{v})$ "1" sein muß. Es gilt also:

$$\mathcal{P}_1(\underline{u}) \wedge \mathcal{P}_1(\underline{v}) \neq \underline{0}.$$

Sei $(\underline{t}, \underline{u}) \in A$.

$$\begin{aligned} \underline{u} = \text{Ja-Knoten} &\implies \varphi(\underline{u}) = \varphi(\underline{t}) \vee \mathcal{P}_1(\underline{u}) \\ &\implies \varphi(\underline{u}) \wedge \mathcal{P}_1(\underline{v}) \neq \underline{0} \end{aligned} \quad (\text{Def. 2.4.12})$$

$\underline{u}$ = Ja-Knoten $\wedge$ $\underline{v}$ = Ja-Sohn von $\underline{u}$

$\Rightarrow (\varphi(\underline{u}) \wedge \varphi_1(\underline{v}) \neq 0) \wedge (\underline{v} = \text{Ja-Sohn von } \underline{u})$

$\Rightarrow \underline{v} \in \text{UZL}. \quad \square$

(Aussage iv des Satzes 2.4.15)

Lemma 2.4.45

Sei $\underline{v}$ = Ja-Sohn von $\underline{u}$. Sei $\underline{u} \in \text{ZL} \setminus \{\emptyset\}$. Seien $\varphi(\underline{u}), \varphi(\underline{v}) \in F_j$.

$\underline{u}$ = Nein-Knoten $\wedge \exists \underline{s} (\underline{s} \in \text{ZL} \wedge \varphi(\underline{s}) \in F_j \wedge \underline{s} = \text{Vorgänger von } \underline{u} \wedge \underline{s} = \text{Ja-Knoten})$

$\Rightarrow \underline{v} \in \text{UZL}.$

Beweis:

Aus $\varphi(\underline{v}) \in F_j$ und $\varphi(\underline{s}) \in F_j$ folgt:

$$\varphi_1(\underline{s}) \wedge \varphi_1(\underline{v}) \neq 0.$$

Sei $(\underline{x}, \underline{s}) \in A$.

$$\underline{s} = \text{Ja-Knoten} \Rightarrow \varphi(\underline{s}) = \varphi(\underline{x}) \vee \varphi_1(\underline{s}) \quad (\text{Def. 2.4.12})$$

$$\varphi(\underline{s}), \varphi(\underline{u}) \in F_j \wedge \underline{s} = \text{Ja-Knoten} \wedge \underline{s} = \text{Vorgänger von } \underline{u} \wedge \underline{s} \in \text{ZL}$$

$$\Rightarrow \forall \underline{y} (\varphi(\underline{y}) \in F_j \wedge \underline{y} = \text{Vorgänger von } \underline{u} \wedge \underline{y} = \text{Nachf. von } \underline{s} \Rightarrow \underline{y} = \text{Nein-Knoten})$$

$$\Rightarrow \varphi(\underline{s}) = \varphi(\underline{u})$$

$$\Rightarrow \varphi(\underline{u}) \wedge \varphi_1(\underline{v}) \neq 0. \quad (\varphi(\underline{x}) \vee \varphi_1(\underline{s}) \wedge \varphi_1(\underline{v}) \neq 0)$$

Die Aussage dieses Satzes folgt dann aus der Aussage iv des Satzes 2.4.15. $\square$

Satz 2.4.46

Sei $SU = (\text{MAZU}, AS)$ und $SU' = (\text{MARZ}, AS)$. Sei $VP = (\text{VST}, SU, 0)$ und

$VP' = (\text{VST}, SU', 0)$.

$$AW(VP) \geq AW(VP')$$

Beweis:

Aus dem in Bemerkung 2.4.42 angegebenen Grunde folgt, daß VP und VP' die gleiche Anzahl von Schritten zur Markierung der entsprechenden Knoten benötigen, wenn die Bedingungen von Fall 1, von Fall 2.3 und von Fall 3 der Markierungsregel MARZ erfüllt sind.

Für MAZU wird im Falle 2.1 sowohl der Ja-Sohn als auch der Nein-Sohn eines Ausgangsknotens (z.B. $\underline{u}$) markiert, unabhängig davon, ob diese Söhne zulässig oder unzulässig sind.

Für MARZ wird im Falle 2.1 und im Falle 2.2 ein zusätzlicher Test durchgeführt, mit dessen Hilfe im voraus festgestellt werden kann, ob der Ja-Sohn (z.B. $\underline{v}$) des Ausgangsknotens unzulässig ist (Lemma 2.4.44 und Lemma 2.4.45). Ist dies der Fall, so braucht der VP' den Knoten $\underline{v}$, der die Wurzel des auszuschließenden Teilbaums darstellt, nicht zu markieren. Aus diesem Grunde benötigt VP' bei jedem auszuschließenden Teilbaum zwei Schritte weniger als VP, nämlich einen Schritt zu $\underline{v}$ und einen anderen Schritt von $\underline{v}$ zurück nach $\underline{u}$.

Wenn $|F_j| \leq 1$ für $\forall j \in \mathbb{N}_m$, dann gilt $AW(VP) = AW(VP')$. In diesem Falle können die Bedingungen von Fall 2.1 der Markierungsregel MARZ niemals erfüllt werden. $\square$

Satz 2.4.47

Sei $SU = (MR, AS)$ und $SU' = (MR, AS')$. Sei $MR \in \{MAZU, MARZ\}$.

Sei $VP = (VST, SU, 0)$ und $VP' = (VST, SU', 0)$.

$$AS' \geq AS \implies AW(VP) \geq AW(VP').$$

Beweis:

Seien $t, t' \in \mathbb{R}^+$. Sei $[0, t]$ die Laufphase von VP und $[0, t']$ die Laufphase von VP'. Sei MBL die Menge aller markierten zulässigen Blätter von VST. Sei US (US') die untere Schranke von VP (VP'), für die gilt:

$$US = \begin{cases} \max\{\omega(\underline{v}) \mid \underline{v} \in MBL\}, & \text{wenn } MBL \neq \emptyset. \\ AS, & \text{wenn } MBL = \emptyset. \end{cases}$$

$$US' = \begin{cases} \max\{\omega(\underline{v}) \mid \underline{v} \in MBL\}, & \text{wenn } MBL \neq \emptyset. \\ AS', & \text{wenn } MBL = \emptyset. \end{cases}$$

Sei $t_1 \in \mathbb{R}^+$ ($t'_1 \in \mathbb{R}^+$) der Zeitpunkt, zu dem zum ersten Male VP (VP') ein zulässiges Blatt $\underline{u}$ markiert, für das gilt: $\omega(\underline{u}) \geq AS'$.

- i) In Zeitintervall $[t_1, t]$ hat VP die gleiche untere Schranke wie VP' in Zeitintervalle $[t'_1, t']$. Da VP und VP' die gleiche Markierungsregel benutzen, machen sie in diesen Zeitintervallen die gleiche Anzahl von Schritten.

- ii) In $[0, t_1)$ gilt für VP' die untere Schranke $US' = AS'$, da kein zulässiges Blatt markiert werden kann, bevor $\underline{u}$ zum Zeitpunkt t_1 markiert ist. Für VP gilt in $[0, t_1)$ die untere Schranke $US = AS \leq AS'$. Deshalb besteht die Möglichkeit, daß VP in $[0, t_1)$ mehrere zulässige Blätter markiert, bevor er den Übergang zu Blatt $\underline{u}$ macht. Dies bedeutet, daß die Anzahl der Schritte von VP in $[0, t_1) \geq$ die Anzahl der Schritte von VP' in $[0, t_1)$ sein muß.

Aus i) und ii) folgt: $AW(VP) \geq AW(VP')$. $\square$

Bemerkung 2.4.48

Die spezielle Struktur der E-Liste, auf die sich die Verzweigungsstruktur bezieht, gibt dem binären Verzweigungsprozeß die Möglichkeit, sich folgendermaßen zu verhalten:

- i) Eine möglichst große untere Schranke soll bei der Markierung des ersten Blattes der binären Verzweigungsstruktur aufgefunden werden.
- ii) Der Aufwand soll reduziert werden, wenn die Markierungsregel MARZ oder eine größere Anfangsschranke benutzt wird.

Beispiel 2.4.49

Dieses Beispiel ist bis auf $SU' = (MARZ, 0)$ identisch mit dem Beispiel 2.4.36. In Bild 2.4.3 wird der binäre Verzweigungsbaum $T = (P, A)$ dargestellt, bei dem ausschließlich diejenigen Knoten gezeigt werden, die sich nach einer Tour des binären Verzweigungsprozesses $VP' = (VST, SU', 0)$ im Zustand "inaktiv" befinden.

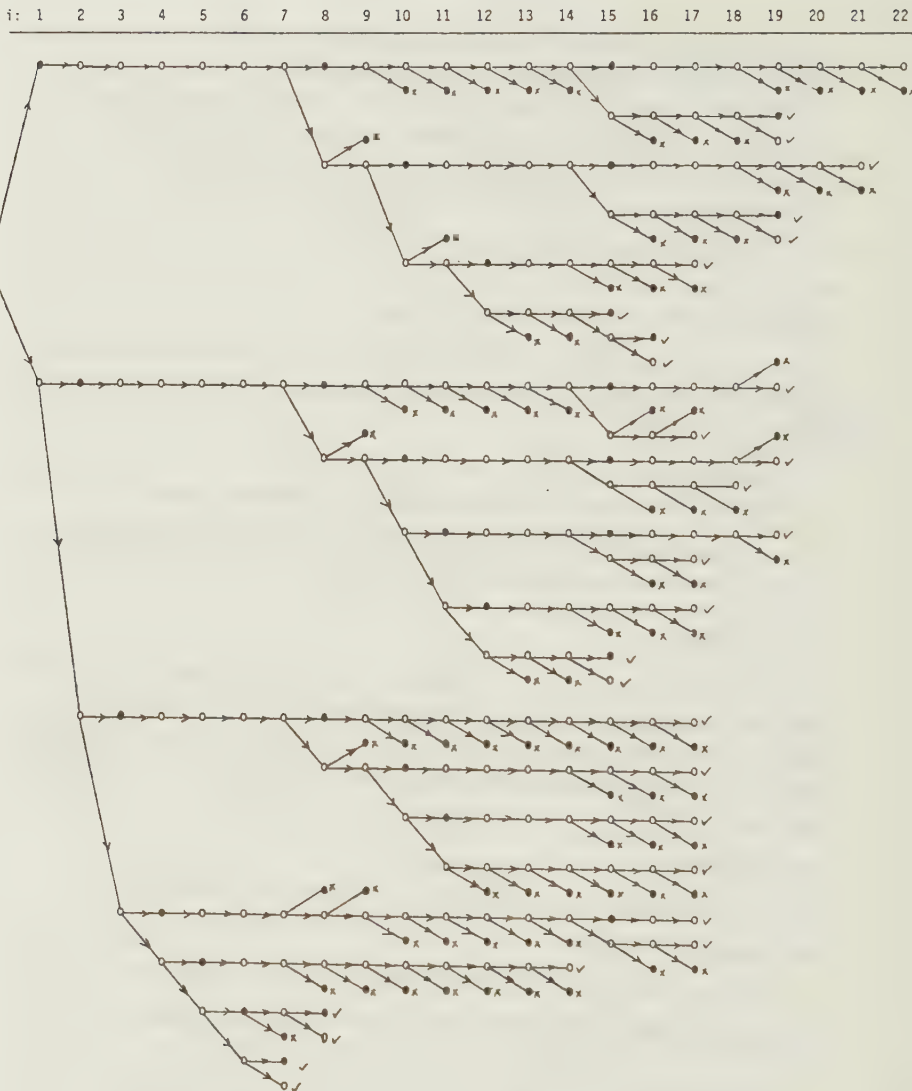
Das Ergebnis:

Das maximale Blatt: $\underline{v} = (1,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0)$.

Das Maximum: $\hat{Z} = 165$.

Der Aufwand: 288 Knoten (Zum Vergleich: 352 Knoten in Beispiel 2.4.36).

Das maximale Blatt wird bereits bei einem Aufwand von 32 Knoten markiert.



$$I_{\underline{v}} = \{1, 8, 15\}.$$

$$\hat{Z} = \omega(\underline{v}) = 165.$$

○ Hein-Knoten

● Ja-Knoten

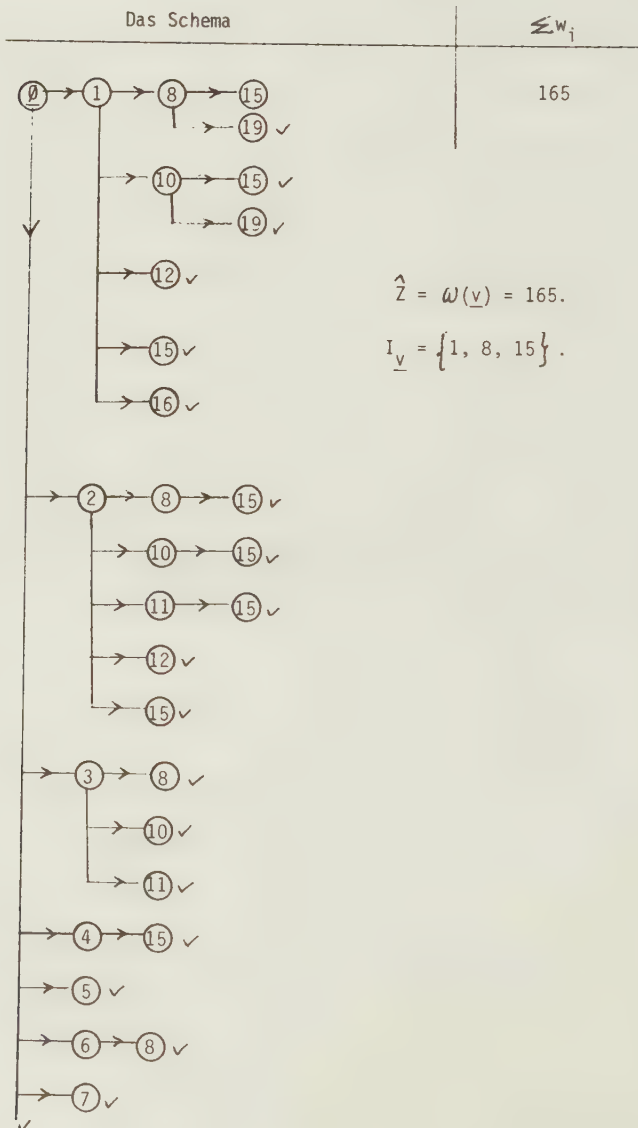
Bild 2.4.3

Bemerkung 2.4.50

Der binäre Verzweigungsbaum mit den markierten Knoten liefert ein anschauliches Schema zur Darstellung der Rechenprozedur, die von einem binären Verzweigungsprozeß im Rechensystem bei seiner Tour durchgeführt wird. Das Schema zeigt, wie der binäre Verzweigungsbaum im Laufe der Tour "heranwächst" und wie der binäre Verzweigungsprozeß Schritt für Schritt die einzelnen Knoten markiert. Das Schema ist jedoch ungeeignet, um Probleme manuell, mit Bleistift und Papier allein, zu lösen. In Bild 2.4.4 wird eine systematische Methode gezeigt, wie man mit wenig Aufwand die Rechenprozedur des binären Verzweigungsprozesses manuell darstellen kann. Dazu wird das Beispiel 2.4.49 verwendet. Man geht von einer vorliegenden E-Liste aus und baut bei jedem vorwärtsgang Schritt für Schritt einen Zweig (in die horizontale Richtung) aus. Für jeden Zweig werden ausschließlich die zulässigen Ja-Knoten angegeben; die Nein-Knoten erscheinen nicht im Schema. Wird der Zweig vollständig, so erhält man ein zulässiges Blatt und eventuell eine neue untere Schranke. Besitzt der Zweig einen Knoten $\underline{v}$, welcher der Bedingung $\omega(\underline{v}) + \Phi(\underline{v}) \leq US$ genügt, so bildet $\underline{v}$ den Endknoten des Zweiges und $\underline{v}$ wird mit $\checkmark$ gekennzeichnet. In diesem Falle kann der Zweig nicht vollständig werden. Man geht dann rückwärts und versucht, bei demselben Zweig einen Nein-Knoten zu finden, der mit "R" markiert ist. Gibt es einen solchen Knoten, dann wird von diesem Knoten aus ein neuer Zweig gebildet. Gibt es ihn nicht, dann ist die Wurzel des binären Verzweigungsbaums bereits inaktiv und die Prozedur wird beendet. Die Lösung des Teilproblems P2 ergibt sich dann aus der zuletzt erhaltenen unteren Schranke und den entsprechenden maximalen Blättern.

Bild 2.4.4

Die schematische Darstellung des binären Verzweigungsprozesses zum manuellen Rechnen.



2.5 Der Erweiterungsprozeß

In diesem Abschnitt werden im wesentlichen die folgenden zwei Punkte untersucht:

- i) Der Beweis für die Behauptung, daß die E-Liste Blockindikatoren (kurz, BIR) enthält, die für die Bestimmung des Maximums belanglos sind. Solche "redundanten" BIR können von der E-Liste ausgeschlossen werden, ohne das Ergebnis des binären Verzweigungsprozesses zu beeinflussen. Die in der E-Liste übrig gebliebenen BIR werden als "wesentlich" bezeichnet.
- ii) Die Entwicklung eines Rechenprozesses, der in der Lage sein soll, alle wesentlichen BIR direkt aus den gegebenen Parametern V , g und f aufzufinden.

Die Ausgangsstellung dieser Untersuchung ist die Erkenntnis, daß ein Block im Grunde nicht schlicht als eine Teilmenge von V , sondern als ein Teilgraph des Netzes NG (nach Def. 2.1.6 Blockteilgraph genannt) betrachtet werden sollte. Auf dieser Betrachtungsweise beruhend werden die Blöcke einer zulässigen Zerlegung folgendermaßen klassifiziert:

- i) Der Block enthält einen Blockteilgraphen, der ausschließlich aus isolierten Knoten besteht.
- ii) Der Blockteilgraph besteht aus einer einzigen Zusammenhangskomponente (kurz, Z-Komponente) von $a \in V$, wobei a kein isolierter Knoten des Blockteilgraphen ist.
- iii) Der Blockteilgraph besteht aus mindestens zwei Z-Komponenten.

Die unter i) klassifizierten Blöcke sind für die weitere Betrachtung uninteressant, da sie den Wert Null besitzen und für sie keine BIR gebildet werden können. Die Knotenmenge des Blockteilgraphen, der unter iii) klassifiziert ist, läßt sich weiter zerlegen, sodaß "kleinere" Blöcke entstehen, deren Blockteilgraphen entweder unter i) oder unter ii) klassifiziert werden können. Führt man diese Zerlegung für alle Blöcke der Klasse iii) durch, dann bleiben am Ende nur diejenigen Blöcke übrig, deren Blockteilgraphen Z-Komponenten sind. Dies ist die Hauptidee für den Beweis, daß alle BIR der unter i) oder iii) klassifizierten Blöcke "redundant" sind.

Definition 2.5.1

Sei $NG = (V, E, g, f)$ ein ungerichtetes Netz und $b \in V$ ein Block.

Ein Element $h \in b$ heit genau dann Hauptknoten von b , wenn h das maximale Element der linearen Ordnung (b, VOR) ist.

Die Menge aller Blcke, welche denselben Hauptknoten besitzen, heit Blockfamilie. BF_h bezeichnet die Blockfamilie, deren Blcke den Hauptknoten h besitzen. h ist der Hauptknoten von BF_h .

Sei $b \in BF_h$. Sei $BG = (b, E_b)$ der Blockteilgraph von b . Dann wird h als Hauptknoten von BG bezeichnet.

Sei $F_h = \{(\underline{s}, w(\underline{s})) \mid (\underline{s}, w(\underline{s})) = \text{BIR von } b \in BF_h\}$. Dann heit F_h die zugeordnete Indikatorfamilie von BF_h (kurz, BIR-Familie).

Folgerung 2.5.2

- i) $|F_h| \leq |BF_h|$
- ii) Der Hauptknoten einer Blockfamilie bildet den Familienindex der zugeordneten Indikatorfamilie.

Definition 2.5.3

Sei BF_h eine Blockfamilie. Sei $GB = (b, E_b)$ der Blockteilgraph von $b \in BF_h$.

b heit einfach genau dann, wenn GB eine Z-Komponente von h in GB ist.

Besteht GB aus mindestens 2 Z-Komponenten, dann heit b mehrfach.

b heit wesentlich genau dann, wenn b einfach ist und $|b| \geq 2$. Sei $BW_h \subseteq BF_h$ die Menge aller wesentlichen Blcke von BF_h . b heit redundant genau dann, wenn $b \in BF_h \setminus BW_h$.

Der Blockindikator (kurz, BIR) eines einfachen (mehrfachen, wesentlichen, redundanten) Blocks, dessen Wert > 0 ist, wird einfacher (mehrfacher, wesentlicher, redundanter) BIR genannt.

Sei F_h eine BIR-Familie. $K_h \subseteq F_h$ heit Kern von F_h genau dann, wenn gilt:

$$K_h = \{(\underline{s}, w(\underline{s})) \mid (\underline{s}, w(\underline{s})) \in F_h \wedge (\underline{s}, w(\underline{s})) \text{ ist einfach}\}.$$

Folgerung 2.5.4

- i) b ist wesentlich $\Leftrightarrow b$ ist einfach und $w(b) > 0$.
- ii) b ist mehrfach $\Rightarrow b$ ist redundant.
- iii) $w(b) = 0 \Rightarrow b$ ist redundant.
- iv) b ist redundant und $w(b) > 0 \Rightarrow b$ ist mehrfach.
- v) $(\underline{s}, w(\underline{s})) \in K_h \Leftrightarrow (\underline{s}, w(\underline{s}))$ ist wesentlich.
- vi) $(\underline{s}, w(\underline{s})) \in F_h \setminus K_h \Leftrightarrow (\underline{s}, w(\underline{s}))$ ist ein redundanter BIR.
- vii) $|K_h| \leq |F_h|$.

Satz 2.5.5

Sei $NG = (V, E, g, f)$ ein ungerichtetes Netz und a die Gewichtsgrenze.
 Sei VOR die Knotenrelation auf V . Sei BF_i eine Blockfamilie mit dem
 Hauptknoten $i \in V$. Sei $b \in BF_i$ ein einfacher Block und $GB = (b, E_b)$ der
 Blockteilgraph von b . Sei $j \in V \setminus b$. Sei $b' = b \cup \{j\}$.

- i) $(i, j) \in VOR \wedge \sum_{l \in b'} g_l \leq a \Leftrightarrow b' \in BF_i$.
- ii) Es gibt einen Pfad, der i mit j verbindet $\Leftrightarrow b'$ ist wesentlich.

Beweis:

- i) $\sum_{l \in b'} g_l \leq a \Leftrightarrow b'$ ist ein Block. (Def. 2.1.6)

Es bleibt noch zu beweisen, daß

$$(i, j) \in VOR \Leftrightarrow b' \in BF_i.$$

" $\Rightarrow$ "

$$(i, j) \in VOR$$

$$\Rightarrow i \text{ ist das maximale Element von } (b', VOR)$$

$$\Rightarrow i \text{ ist der Hauptknoten von } b' \quad (\text{Def. 2.5.1})$$

$$\Rightarrow b' \in BF_i.$$

" $\Leftarrow$ "

$$b' \in BF_i$$

$$\Rightarrow i \text{ ist das maximale Element von } (b', VOR) \quad (\text{Def. 2.5.1})$$

$$\Rightarrow (i, j) \in VOR.$$

ii) " $\Rightarrow$ "

Es gibt einen Pfad, der i mit j verbindet

$\Rightarrow GB' = (b', E_{b'})$ mit $E_{b'} = b' \times b' \cap E$ ist eine Z-Komponente von i in GB'

$\Rightarrow b'$ ist einfach und $|b'| \geq 2$

$\Rightarrow b'$ ist wesentlich. (Def. 2.5.3)

" $\Leftarrow$ " Offensichtlich. $\square$

Bemerkung 2.5.6

Die Aussagen i) und ii) des Satzes 2.5.5 sind wichtige Kriterien zur Auffindung aller wesentlichen Blöcke, die denselben Hauptknoten besitzen. Die Aussage i) wird als Kriterium zum Test der Gewichtsgrenze betrachtet. Mit Hilfe der Aussage ii) wird festgestellt, ob der neugebildete Block wesentlich ist. Der noch anzugebende Erweiterungsprozeß benutzt diese beiden Kriterien, um alle wesentlichen Blöcke einer Blockfamilie aufzufinden. Der Kern der entsprechenden BIR-Familie ergibt sich unmittelbar aus dem Ergebnis des Erweiterungsprozesses.

Wir müssen nun noch die Behauptung begründen, daß allein die Betrachtung der Menge aller wesentlichen Blöcke hinreichend ist, um das Problem P2 zu lösen. D.h. es muß nachgewiesen werden, daß dasselbe Ergebnis erzielt wird, wenn sich die binäre Verzweigungsstruktur auf eine E-Liste bezieht, die ausschließlich aus den Kernen der BIR-Familien (anstatt der BIR-Familien selbst) besteht. Da die Mächtigkeit des Kerns i.a. sehr viel kleiner ist als die der entsprechenden BIR-Familien, erhält man als Ergebnis der Phase 1 eine bedeutend kürzere E-Liste. Dies ist gleichbedeutend mit der starken Reduzierung des Aufwands des binären Verzweigungsprozesses.

Die Beweisidee beruht im wesentlichen auf der Erkenntnis, daß ein redundanter Block, der mehr als einen Knoten enthält, sich in mehrere einfache Blöcke zerlegen läßt. Einer von den sich daraus ergebenden Blöcken ist wieder ein einfacher Block mit demselben Hauptknoten; die übrigen einfachen Blöcke haben eigene, voneinander unterscheidbare Hauptknoten. Einfache Blöcke, die redundant sind, lassen sich von der weiteren Betrachtung ausschließen, da für sie keine BIR gegeben werden können. So bleiben von der Zerlegung nur wesentliche Blöcke übrig, deren BIR jeweils einer entsprechenden BIR-Familie

zugeordnet werden. Enthält eine BIR-Familie einen BIR, der mit dem neuen, aus der Zerlegung sich ergebenden identisch ist, dann bleibt der neue außer Betracht. Da dies nach der Definition der BIR-Familie der Fall sein muß, so bleibt schließlich keiner von den aus der Zerlegung sich ergebenden einfachen Blöcken übrig. Dies bedeutet, daß ein redundanter BIR aus der BIR-Familie, zu der er gehört, entfernt werden darf, ohne das Maximum der binären Verzweigungsstruktur zu beeinträchtigen. (Deshalb wird der Block als redundant bezeichnet). Dieser "Zerlegungsprozeß" läßt sich für alle noch vorhandenen redundanten BIR der BIR-Familien so lange durchführen, bis schließlich nur die wesentlichen BIR in der E-Liste bleiben. Diese besteht nun ausschließlich aus den Kernen der BIR-Familien. Eine binäre Verzweigungsstruktur, die sich auf diese neue E-Liste bezieht, besitzt dasselbe Maximum wie die auf die ursprüngliche E-Liste bezogene binäre Verzweigungsstruktur.

Satz 2.5.7

Sei BF_h eine Blockfamilie. Sei F_h die BIR-Familie von BF_h und K_h der Kern von F_h .

- i) $K_h = F_h \iff$ alle $b \in BF_h \setminus \{h\}$ sind wesentlich.
- ii) $K_h = \emptyset \iff$ alle $b \in BF_h$ sind redundant $\iff$ alle BIR von F_h sind redundant.

Beweis: Die beiden Aussagen ergeben sich aus der Definition 2.5.3. $\square$

Bemerkung 2.5.8

Der Fall $K_h = \emptyset$ tritt genau dann ein, wenn h ein isolierter Knoten in allen Blockteilgraphen mit dem Hauptknoten h ist. Als Beispiel für den Fall $K_h = F_h$ betrachten wir die Blockfamilie BF_h , in der alle Blöcke, die mehr als einen Knoten enthalten, die folgende Eigenschaft besitzen: Sei $b \in BF_h \setminus \{h\}$, dann existiert für jeden Knoten $x \in b$ mindestens ein Pfad, der x mit h verbindet.

Definition 2.5.9

Sei BF_h eine Blockfamilie. Sei $b \in BF_h$. Sei $GB = (b, E_b)$ der Blockteilgraph von b .

$\tilde{b} = \{b_i \mid i \in \mathbb{N}_k\}$ heißt Zerlegung von b genau dann, wenn gilt:

$$\forall i: b_i \neq \emptyset$$

$$\bigcup_{i \in \mathbb{N}_k} b_i = b$$

$$\forall i, j (i \neq j \Rightarrow b_i \cap b_j = \emptyset)$$

$$\forall b_i \exists h' (h' = \text{Hauptknoten von } b_i \Rightarrow G_i = Z\text{-Komponente von } h' \text{ in } G_i).$$

Dabei ist $G_i = (b_i, E_i)$ der Blockteilgraph von b_i , für den gilt: $E_i = b_i \times b_i \cap E_B$.

b_i heißt Unterblock von b . Gilt $h = h'$, dann heißt der Unterblock Kern von b .

Eine Zerlegung heißt echt, wenn $|\pi_b| \geq 2$. b heißt zerlegbar genau dann, wenn es eine echte Zerlegung von b gibt. Existiert keine echte Zerlegung von b , dann heißt b unzerlegbar.

Folgerung 2.5.10

- i) Es gibt genau eine Zerlegung von b .
- ii) b ist zerlegbar $\Leftrightarrow b$ ist mehrfach.
- iii) b ist unzerlegbar $\Leftrightarrow b$ ist einfach $\Leftrightarrow b = b_i$.
- iv) b ist wesentlich $\Leftrightarrow b$ ist unzerlegbar und $w(b) > 0$.
- v) Jeder Unterblock ist einfach.
- vi) b_i ist wesentlich $\Leftrightarrow |b_i| \geq 2 \Rightarrow w(b_i) > 0$.
- vii) $w(b) = \sum_{i \in \mathbb{N}_k} w(b_i)$.
- viii) Sei b ein zerlegbarer Block und GB dessen Blockteilgraph. Sei b_i der Kern von b . Sei $rb = b \setminus b_i$.
 $w(b) = w(b_i) \Leftrightarrow rb$ enthält ausschließlich isolierte Knoten von GB.

Satz 2.5.11

Sei b ein zerlegbarer Block. Sei $\pi_b = \{b_i | i \in \mathbb{N}_k\}$ die Zerlegung von b .

Sei $(\underline{s}, w(\underline{s}))$ der BIR von b . Sei $(\underline{s}_i, w_i)$ der BIR von b_i für $\forall i \in \mathbb{N}_k$.

- i) $\underline{s} = \bigvee_{i \in \mathbb{N}_k} \underline{s}_i$.
- ii) $\forall i, j \in \mathbb{N}_k (i \neq j \Rightarrow \underline{s}_i \wedge \underline{s}_j = \underline{0})$.
- iii) $w(\underline{s}) = \sum_{i \in \mathbb{N}_k} w_i$.

Beweis:

Die Aussagen i) und ii) ergeben sich aus der Definition 2.5.9.

Die Aussage iii) folgt aus der Aussage vii) der Folgerung 2.5.10. $\square$

Satz 2.5.12

Sei BF_h eine Blockfamilie und F_h die BIR-Familie. Sei K_h der Kern von F_h . Sei $(\underline{x}, w(\underline{x})) \in F_h$ der BIR von $b \in BF_h$. Sei $(\underline{y}, w(\underline{y}))$ der BIR eines Unterblocks b_j von b .

i) b ist wesentlich $\iff (\underline{x}, w(\underline{x})) = (\underline{y}, w(\underline{y}))$.

ii) $b_i = \text{Kern von } b \wedge |b_i| \geq 2 \implies (\underline{y}, w(\underline{y})) \in K_h$

iii) Sei $j \in V$ und $j \neq h$. Sei F_j eine BIR-Familie. Sei K_j der Kern von F_j .

$$(\underline{y}, w(\underline{y})) \in F_j \iff (\underline{y}, w(\underline{y})) \in K_j$$

Beweis:

i) b ist wesentlich $\iff b$ ist unzerlegbar und $w(b) > 0$. (Folg. 2.5.10iv)

$$\iff b = b_i \text{ und } w(b) = w(b_i) > 0. \quad (\text{Folg. 2.5.10iii})$$

$$\iff (\underline{x}, w(\underline{x})) = (\underline{y}, w(\underline{y})).$$

ii) $b_i = \text{Kern von } b \wedge |b_i| \geq 2 \implies (\underline{y}, w(\underline{y})) \in F_h \wedge$ (Def. 2.5.9,

$(\underline{y}, w(\underline{y}))$ ist einfach (Folg. 2.5.10)

$$\iff (\underline{y}, w(\underline{y})) \in K_h. \quad (\text{Def. 2.5.3})$$

iii) Die Aussage ergibt sich aus der Definition 2.5.3, da ein Unterblock einfach sein muß. $\square$

Definition 2.5.13

Sei $F = \bigcup_{j \in V} F_j$ eine Indikatormenge. Sei K_j der Kern von F_j für $\forall j \in V$.

$KF = \bigcup_{j \in V} K_j$ heit Kern von F . Sei $|KF| = 1$.

Seien $L_i = (F_i, R_f, \alpha_f)$, $L_j = (F_j, R_f, \alpha_f)$. Sei $(\underline{x}, w(\underline{x})) \in K_i \subseteq F_i$ und $(\underline{y}, w(\underline{y})) \in K_j \subseteq F_j$.

Eine K-Liste ist ein Tripel $KL = (KF, R_k, \alpha_k)$, wobei folgendes gilt:

i) $R_k \subseteq KF \times KF$, angegeben durch:

$$(\underline{x}, w(\underline{x})) R_k (\underline{y}, w(\underline{y})) \iff (\alpha_1(L_i) < \alpha_1(L_j)) \vee \\ (\alpha_1(L_i) = \alpha_1(L_j) \wedge \alpha_f(\underline{x}, w(\underline{x})) \leq \alpha_f(\underline{y}, w(\underline{y})))$$

ii) $\alpha_k : KF \xrightarrow{\text{bij}} \mathbb{N}_1$, angegeben durch:

$$\alpha_k(\underline{x}, w(\underline{x})) \leq \alpha_k(\underline{y}, w(\underline{y})) \iff (\underline{x}, w(\underline{x})) R_k (\underline{y}, w(\underline{y}))$$

$\alpha_k(\underline{x}, w(\underline{x}))$ bzw. $\alpha_k(\underline{y}, w(\underline{y}))$ heißt Listennummer von $(\underline{x}, w(\underline{x}))$ bzw. von $(\underline{y}, w(\underline{y}))$ in KL. 1 heißt Länge der K-Liste.

Bemerkung 2.5.14

$R_k \subseteq R_e$ (Def. 2.3.35) ist eine lineare Ordnung auf KF. Eine K-Liste läßt sich aus einer vorliegenden E-Liste bilden, indem man alle redundanten BIR von der E-Liste entfernt. Diese Methode ist leider sehr aufwendig, da zunächst die E-Liste aufgefunden werden muß und die Anzahl der in ihr enthaltenen BIR exponentiell mit $|V|$ zunehmen kann. Die Aufgabe des noch anzugebenden Erweiterungsprozesses besteht darin, KF statt F mittels der in Satz 2.5.5 angegebenen Kriterien aufzufinden. KL ergibt sich dann aus KF mit Hilfe der Prozedur SORT.

Satz 2.5.15

Sei F eine Indikatormenge und KF der Kern von F. Sei $|F| = m$ und $|KF| = 1$. Sei $EL = (F, R_e, \alpha_e)$ und $KL = (KF, R_k, \alpha_k)$. Sei $VST_1 = (P_1, A_1, F, p_1, \omega_1, \varphi_1, \bar{x}_1)$ ($VST_2 = (P_2, A_2, KF, p_2, \omega_2, \varphi_2, \bar{x}_2)$) eine binäre Verzweigungsstruktur bezüglich EL (KL). Sei ZB_1 (ZB_2) die Menge der zulässigen Blätter von VST_1 (VST_2).

i) Sei $\underline{v} \in ZB_1$. Es gibt genau ein $\underline{u} \in ZB_2$, sodaß $\omega_2(\underline{u}) = \omega_1(\underline{v})$ und $I_{\varphi_2(\underline{u})} \subseteq I_{\varphi_1(\underline{v})}$.

ii) Sei $\underline{u} \in ZB_2$. Es gibt mindestens ein $\underline{v} \in ZB_1$, sodaß $\omega_1(\underline{v}) = \omega_2(\underline{u})$ und $I_{\varphi_1(\underline{v})} \supseteq I_{\varphi_2(\underline{u})}$.

iii) Seien $\underline{v} \in ZB_1$, $\underline{u} \in ZB_2$ und es gelte $\omega_1(\underline{v}) = \omega_2(\underline{u})$, $I_{\varphi_1(\underline{v})} \supseteq I_{\varphi_2(\underline{u})}$.
 $\underline{v}$ ist maximal $\iff \underline{u}$ ist maximal.

Beweis:

i) Sei $\underline{v} = (v_1, \dots, v_m) \in \{0, 1\}^m$. Sei $(s_i, w_i) \in F$ für $\forall i \in \mathbb{N}_m$.

$$\omega_1(\underline{v}) = \sum_{i \in \mathbb{N}_m} v_i w_i. \quad (\text{Satz 2.4.13i})$$

$$\varphi_1(\underline{v}) = \bigvee_{i \in \mathbb{N}_m} v_i s_i, \quad (\text{Satz 2.4.13iii})$$

$$\forall i, j \in \mathbb{N}_m (i \neq j \Rightarrow v_i s_i \wedge v_j s_j = \underline{0}). \quad (\text{Def. 2.4.14i})$$

Es ist also zu beweisen, daß für $\underline{v} \in \text{ZB}_1$ ein $\underline{u} \in \text{ZB}_2$ mit $\underline{u} = (u_1, \dots, u_1)$ existiert, sodaß $\underline{u}$ und $\underline{v}$ folgenden Relationen genügen ($(\underline{s}_j', w_j') \in \text{KF}$):

$$\omega_2(\underline{u}) = \sum_{j \in \mathbb{N}_1} u_j w_j' = \omega_1(\underline{v}). \quad (1)$$

$$I\varphi_2(\underline{u}) \subseteq I\varphi_1(\underline{v}), \text{ wobei } \varphi_2(\underline{u}) = \bigvee_{j \in \mathbb{N}_1} u_j s_j'. \quad (2)$$

$$\forall i, j \in \mathbb{N}_1 (i \neq j \Rightarrow u_i s_i' \wedge u_j s_j' = \underline{0}). \quad (3)$$

Es wird zunächst gezeigt, daß für jedes $\underline{v} \in \text{ZB}_1$ ein $\underline{u} \in \text{ZB}_2$ konstruiert werden kann, das den Bedingungen (1), (2) und (3) genügt. Anschließend wird bewiesen, daß es kein anderes Blatt von VST_2 gibt, das (1), (2) und (3) genügt.

Um die Bedingungen (1), (2) und (3) zu testen, braucht man ausschließlich diejenigen $(\underline{s}_i, w_i) \in F$ zu betrachten, für die gilt: $v_i = 1$. Für jedes i , bei dem $v_i = 1$ gilt, gibt es ein $h \in V$, sodaß $(\underline{s}_i, w_i) \in F_h \subseteq F$. Sei K_h der Kern von F_h . Für K_h bestehen die folgenden zwei Möglichkeiten:

$$A) K_h = \emptyset.$$

$$B) K_h \neq \emptyset.$$

Fall A

Nach Aussage ii) des Satzes 2.5.7 enthält F_h ausschließlich redundante BIR. Jedes $(\underline{s}_i, w_i) \in F_h$, bei dem $v_i = 1$ gilt, ist der BIR eines zerlegbaren Blocks, dessen Hauptknoten h ein isolierter Knoten ist (Bemerkung 2.5.8). Wird der Block zerlegt, dann erhält man mindestens einen BIR, der den Aussagen i), ii) und iii) des Satzes 2.5.11 genügt. Alle derart entstehenden BIR sind wesentlich, da die Unterblöcke einfach sein müssen (Folg. 2.5.10v). Seien $(\underline{s}_{i1}, w_{i1}), \dots, (\underline{s}_{it}, w_{it})$ die BIR der sich bei der Zerlegung ergebenden wesentlichen Unterblöcke. Sei $k \in \mathbb{N}_t$. Nach Aussage iii) des Satzes 2.5.12 gilt $(\underline{s}_{ik}, w_{ik}) \in F_e$, $e \neq h$, genau dann, wenn $(\underline{s}_{ik}, w_{ik}) \in K_e$, wobei K_e der Kern von F_e sein soll. Dies bedeutet, daß für jedes $(\underline{s}_{ik}, w_{ik}) \in F_e$ ein BIR $(\underline{s}_{jk}', w_{jk}') \in \text{KF}$ existiert, sodaß $(\underline{s}_{jk}', w_{jk}') = (\underline{s}_{ik}, w_{ik}) \in K_e \subseteq \text{KF}$ und $jk \in \mathbb{N}_1$.

Für jedes $(\underline{s}'_{jk}, w'_{jk}) \in KF$ wird eine Eins-Komponente von $\underline{u}$, $u_{jk} = 1$, gebildet, wobei $jk \in \mathbb{N}_1$ und $k \in \mathbb{N}_t$. Dementsprechend gibt es eine Null-Komponente von $\underline{v}$: $v_{ik} = 0$. Diese Konstruktion, die von $v_i = 1$ ausgeht, läßt sich folgendermaßen darstellen:

$$\underline{v} : \quad v_i = 1 \quad v_{i1} = \dots = v_{ik} = \dots = v_{it} = 0 \quad (ik \in \mathbb{N}_m \text{ für } \forall k \in \mathbb{N}_t)$$

$$\underline{u} : \quad u_{j1} = \dots = u_{jk} = \dots = u_{jt} = 1 \quad (jk \in \mathbb{N}_1 \text{ für } \forall k \in \mathbb{N}_t)$$

Dabei gelten für alle $(\underline{s}_{ik}, w_{ik}) \in F$ und $(\underline{s}'_{jk}, w'_{jk}) \in KF$ die folgenden Relationen:

$$\exists e \neq h : ((\underline{s}_{ik}, w_{ik}) \in F_e \subseteq F \implies (\underline{s}'_{jk}, w'_{jk}) \in K_e \subseteq KF) \quad (\{h\} \text{ ist redundant})$$

$$\sum_{\substack{k \in \mathbb{N}_t \\ jk \in \mathbb{N}_1}} w'_{jk} = \sum_{\substack{k \in \mathbb{N}_t \\ ik \in \mathbb{N}_m}} w_{ik} = w_i \quad (\text{Satz 2.5.11iii})$$

$$\bigvee_{\substack{k \in \mathbb{N}_t \\ jk \in \mathbb{N}_1}} \underline{s}'_{jk} = \bigvee_{\substack{k \in \mathbb{N}_t \\ ik \in \mathbb{N}_m}} \underline{s}_{ik} = \underline{s}_i \quad (\text{Satz 2.5.11i})$$

$$\forall p, q \in \mathbb{N}_t (p \neq q \implies \underline{s}'_{jp} \wedge \underline{s}'_{jq} = \underline{s}_{ip} \wedge \underline{s}_{iq} = \underline{0}) \quad (\text{Satz 2.5.ii})$$

Fall B: $K_h \neq \emptyset$

Dieser Fall läßt sich folgendermaßen unterteilen:

$$B1) \quad K_h = F_h.$$

$$B2) \quad K_h \subsetneq F_h.$$

Fall B1: $K_h = F_h$.

Nach Aussage i) des Satzes 2.5.7 gilt $K_h = F_h$ genau dann, wenn alle BIR von F_h wesentlich sind. Dies bedeutet, daß es für jedes $(\underline{s}_i, w_i) \in F_h$, bei dem $v_i = 1$ gilt, genau ein $(\underline{s}'_j, w'_j) \in K_h$ gibt, sodaß $u_j = 1$. Daraus folgt:

$$w'_j = w_i. \quad (1')$$

$$\underline{s}'_j = \underline{s}_i. \quad (2')$$

Fall B2: $K_h \subsetneq F_h$.

F_h läßt sich in diesem Falle in zwei Teilmengen aufteilen:

$$F'_h = K_h.$$

$$F''_h = F_h \setminus K_h.$$

Wenn $v_i = 1$ und $(s_i, w_i) \in F'_h$, dann ist (s_i, w_i) wesentlich. Dieser Fall ist identisch mit dem Fall B1.

Wenn $v_i = 1$ und $(s_i, w_i) \in F''_h$, dann ist (s_i, w_i) redundant (Folg. 2.5.4 vi). Man erhält wieder den Fall A.

Es bestehen also zwischen den Komponenten in $\underline{v}$ und den Komponenten in $\underline{u}$ die folgenden Zusammenhänge:

Ist $v_j = 1$ und $(s_j, w_j) \in F$ redundant, dann gibt es ein $t \in \mathbb{N}$, sodaß für $\underline{u}$ t Eins-Komponenten, $u_{jt} = - = u_{jt} = 1$, konstruiert werden können. Ist $v_j = 1$ und $(s_j, w_j) \in F$ wesentlich, dann läßt sich genau eine Eins-Komponente von $\underline{u}$, $u_j = 1$, konstruieren.

Auf diese Weise kann man die Eins-Komponenten von $\underline{v}$ im einzelnen betrachten und die entsprechende(n) Eins-Komponente(n) von $\underline{u}$ konstruieren, bis schließlich alle Eins-Komponenten von $\underline{v}$ berücksichtigt worden sind. Der sich daraus ergebende Vektor $\underline{u}$ ist ein Blatt von VST_2 , das den Relationen (1), (2) und (3) genügt.

Es bleibt noch zu beweisen, daß es kein anderes Blatt, z.B. $\underline{u}' \neq \underline{u}$, von VST_2 gibt, das den Relationen (1), (2) und (3) genügt. Angenommen, es gibt ein $\underline{u}' \in ZB_2$, sodaß $\underline{u}' \neq \underline{u} \wedge \omega_2(\underline{u}') = \omega_1(\underline{v}) \wedge I_{\mathcal{H}_2}(\underline{u}') \subseteq I_{\mathcal{H}_1}(\underline{v})$. Nach der eben beschriebenen Konstruktion bedeutet dies, daß mindestens eine Eins-Komponente von $\underline{v}$, z.B. v_k , existiert, für die gilt: $(s_k, w_k) \in F$ ist redundant und (s_k, w_k) ist der BIR eines Blocks, für den mindestens zwei Zerlegungen existieren. Dies steht im Widerspruch zu der Aussage i) der Folgerung 2.5.10.

ii) Sei $\underline{u} = (u_1, \dots, u_1)$. Für $\underline{u}$ kann ein $\underline{v}$ folgendermaßen konstruiert werden:

Man betrachtet die einzelnen Eins-Komponenten von $\underline{u}$. Sei $u_j = 1$. Für u_j gibt es ein $(s'_j, w'_j) \in K_h \subseteq KF$, wobei K_h der Kern von $F_h \subseteq F$ sein soll. Dies bedeutet, daß genau ein $(s_j, w_j) \in F_h$ existiert, sodaß $(s_j, w_j) = (s'_j, w'_j)$. Für jedes (s_j, w_j) , das sich auf diese Weise ergibt, wird eine Eins-Komponente von $\underline{v}$, $v_j = 1$, gebildet. Die anderen Komponenten von $\underline{v}$, für die keine entsprechenden Eins-Komponenten von $\underline{u}$ existieren, bilden die Null-Komponenten von $\underline{v}$. Mit Hilfe dieses Konstruktionsverfahrens (Verfahren ID genannt) wird für jedes $\underline{u} \in ZB_2$ genau ein $\underline{v} \in ZB_1$ festgelegt, bei dem gilt: $\omega_1(\underline{v}) = \omega_2(\underline{u})$ und $I_{\mathcal{H}_1}(\underline{v}) = I_{\mathcal{H}_2}(\underline{u})$.

Es muß noch nachgewiesen werden, daß es möglich ist, für $\underline{u}$ ein $\underline{x} \in \text{ZB}_1$, $\underline{x} \neq \underline{v}$, zu konstruieren, derart, daß $\omega_1(\underline{x}) = \omega_2(\underline{u})$ und $I_{\varphi_1(\underline{x})} \supset I_{\varphi_2(\underline{u})}$.

Sei $u_p = 1$ und $(\underline{s}'_p, w'_p) \in K_e \subseteq F_e$. Man betrachtet für $u_p = 1$ ein $(\underline{s}_q, w_q) \in F_e \setminus K_e$, bei dem gilt:

$$\forall r \in I_{\underline{u}} \setminus \{p\} : (\underline{s}_q \neq \underline{s}'_r \Rightarrow \underline{s}_q \wedge \underline{s}'_r = \underline{0}).$$

Für die Wahl von $(\underline{s}_q, w_q)$, das ein redundanter BIR sein soll (Folg. 2.5.4vi), bestehen zwei Möglichkeiten:

$$a) w_q = w'_p \wedge I_{\underline{s}'_p} \subset I_{\underline{s}_q}.$$

$$b) w_q > w'_p \wedge I_{\underline{s}'_p} \subset I_{\underline{s}_q}.$$

Zu a)

Es wird für $(\underline{s}_q, w_q)$ eine Eins-Komponente von $\underline{x}$, $x_q = 1$, gebildet. Dies ist die einzige Eins-Komponente von $\underline{x}$, für die gilt:

$$(\underline{s}_q, w_q) \neq (\underline{s}'_p, w'_p).$$

Die anderen Eins-Komponenten von $\underline{x}$ werden mittels des Verfahrens ID konstruiert. Auf diese Weise erhält man ein $\underline{x} \neq \underline{v}$, bei dem folgendes gilt:

$$\underline{x} \in \text{ZB}_1 \wedge \omega_1(\underline{x}) = \omega_2(\underline{u}) \wedge I_{\varphi_1(\underline{x})} \supset I_{\varphi_2(\underline{u})}.$$

Zu b)

Aus $w_q > w'_p$ und $(\underline{s}'_p, w'_p) \in K_e$ folgt, daß sich der Block, dessen BIR $(\underline{s}_q, w_q) \in F_e \setminus K_e$ ist, in mindestens zwei Unterblöcke zerlegen läßt. Einer von den Unterblöcken ist ein Block, dessen BIR (z.B. $(\underline{s}_t, w_t)$) identisch ist mit $(\underline{s}'_p, w'_p)$. Von den übrigen Unterblöcken gibt es mindestens einen, der wesentlich ist (Grund: $w_q - w'_p > 0$). Seien $(\underline{s}_{r1}, w_{r1}), \dots, (\underline{s}_{rk}, w_{rk})$ die BIR von diesen wesentlichen Unterblöcken. Dementsprechend existieren $d1 \in V, \dots, dk \in V$, sodaß $(\underline{s}_{r1}, w_{r1}) \in K_{d1} \subseteq F_{d1}, \dots, (\underline{s}_{rk}, w_{rk}) \in K_{dk} \subseteq F_{dk}$. Für $(\underline{s}_q, w_q)$ lassen sich also $(k+1)$ Eins-Komponenten von $\underline{x}$, nämlich $x_t = 1, x_{d1} = 1, \dots, x_{dk} = 1$, konstruieren. Jeder von diesen $(k+1)$ Eins-Komponenten von $\underline{x}$ kann genau eine Eins-Komponente von $\underline{u}$ zugeordnet werden. Für den Rest der Eins-Komponenten von $\underline{u}$ lassen sich genau so viele Eins-Komponenten von $\underline{x}$ mittels des Verfahrens ID konstruieren. Auf diese Weise erhält man ein $\underline{x} \neq \underline{v}$, bei dem gilt: $\underline{x} \in \text{ZB}_1 \wedge \omega_1(\underline{x}) = \omega_2(\underline{u}) \wedge I_{\varphi_1(\underline{x})} \supset I_{\varphi_2(\underline{u})}$.

iii) " $\Rightarrow$ "

Annahme: $\underline{u} \in \text{ZB}_2$ ist nicht maximal.

Daraus folgt, daß $\exists \underline{u}' (\underline{u}' \in \text{ZB}_2 \wedge \omega_2(\underline{u}') > \omega_2(\underline{u}) \wedge \varphi_2(\underline{u}') \neq \varphi_2(\underline{u}))$.

Nach Aussage ii) gibt es für $\underline{u}'$ ($\underline{u}$) mindestens ein $\underline{v}'$ ($\underline{v}$), sodaß $\omega_2(\underline{u}') = \omega_1(\underline{v}')$ ($\omega_2(\underline{u}) = \omega_1(\underline{v})$). Aus $\omega_2(\underline{u}') > \omega_2(\underline{u})$ folgt $\omega_1(\underline{v}') > \omega_1(\underline{v})$. Dies bedeutet, daß $\underline{v} \in \text{ZB}_1$ nicht maximal sein kann. Widerspruch.

" $\Leftarrow$ "

Annahme: $\underline{v} \in \text{ZB}_1$ ist nicht maximal.

Daraus folgt, daß $\exists \underline{v}' (\underline{v}' \in \text{ZB}_1 \wedge \omega_1(\underline{v}') > \omega_1(\underline{v}) \wedge \varphi_1(\underline{v}') \neq \varphi_1(\underline{v}))$.

Nach Aussage i) gibt es für $\underline{v}'$ ($\underline{v}$) genau ein $\underline{u}'$ ($\underline{u}$), sodaß $\omega_2(\underline{u}') = \omega_1(\underline{v}')$ ($\omega_2(\underline{u}) = \omega_1(\underline{v})$). Aus $\omega_1(\underline{v}') > \omega_1(\underline{v})$ folgt $\omega_2(\underline{u}') > \omega_2(\underline{u})$. Dies bedeutet, daß $\underline{u} \in \text{ZB}_2$ nicht maximal sein kann. Widerspruch. $\square$

Bemerkung 2.5.16

Satz 2.5.15 ist der Hauptsatz, in dem begründet wird, daß der binäre Verzweigungsprozeß dasselbe Ergebnis erzielt, wenn sich die entsprechende binäre Verzweigungsstruktur auf die K-Liste (statt auf die E-Liste) bezieht.

Beispiel 2.5.17

Die K-Liste des Beispiels 2.3.45 wird in Bild 2.5.1 angegeben. Bild 2.5.2 zeigt den entsprechenden binären Verzweigungsbaum, der ausschließlich aus denjenigen Knoten besteht, die sich nach einer Tour des binären Verzweigungsprozesses im Zustand "inaktiv" befinden. Die Tour wird gemäß der Suchstrategie MARZ gemacht. Der Aufwand beträgt 48 Knoten. Das Maximum wird bereits bei einem Aufwand von 15 Knoten aufgefunden. Im Vergleich zu Beispiel 2.4.49 bedeutet dies eine sechsfache Aufwandreduzierung.

Das maximale Blatt: $\underline{v} = (1, 0, 0, 1, 0, 1, 0, 0, 0, 0)$.

Das Maximum: $\hat{Z} = 165$.

$\underline{v}$ entspricht den folgenden Blöcken des in Beispiel 2.3.45 angegebenen Netzes:

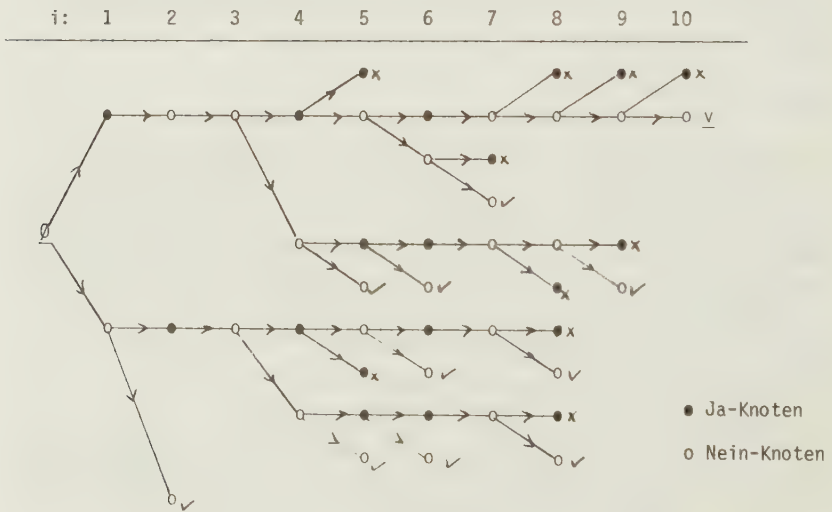
$$\{2, 3\}, \{5, 6\}, \{8, 9, 10\}.$$

Die K-Liste des Beispiels 2.3.45

i	$\underline{s}_i$	w_i
1	0 0 0 0 0 0 0 1 1 0 0 1	80
2	0 0 0 0 0 0 0 1 0 0 0 1	69
3	0 0 0 0 0 0 0 1 1 0 0 0	11
4	0 0 1 0 0 0 0 0 0 0 1 0	50
5	0 0 0 0 0 0 1 0 0 0 1 0	40
6	0 0 0 0 1 0 0 0 0 1 0 0	35
7	0 0 0 0 1 0 0 0 1 0 0 0	10
8	0 0 0 1 0 0 0 0 0 1 0 0	25
9	0 0 0 1 0 0 0 0 0 0 1 0	9
10	0 1 0 0 0 0 0 0 0 0 1 0	1

Bild 2.5.1

Der binäre Verzweigungsbaum (nur markierte Knoten werden gezeigt)



$$\omega(\underline{v}) = \hat{Z} = 165$$

$$\underline{v} = (1,0,0,1,0,1,0,0,0,0)$$

Bild 2.5.2

Für den Rest dieses Abschnitts sei $NG = (V, E, g, f)$ ein ungerichtetes Netz.
Sei a die Gewichtsgrenze. Sei BF_h eine Blockfamilie mit dem Hauptknoten h .

Definition 2.5.18

Sei $b \in BF_h$. Eine Adjazenzstruktur von $b \in BF_h$ ist ein Paar $AS_b \equiv (AZ_b, VOR)$, wobei folgendes gilt:

$$i) AZ_b \equiv \{i \mid i \in V \setminus b \wedge h \text{ VOR } i \wedge \exists j(j \in b \Rightarrow \{i, j\} \in E) \wedge \sum_{l \in b} g_l + g_i \leq a\}$$

heißt Adjazenzmenge von b .

ii) VOR ist die Knotenrelation.

Sei $i \in AZ_b$. Dann heißt $b' \equiv b \cup \{i\}$ die Erweiterung von b .

$EW_b \equiv \{b' \mid b' = b \cup \{i\} \wedge i \in AZ_b\}$ heißt Erweiterungsmenge von b .

Bemerkung 2.5.19

$AS_b = (AZ_b, VOR)$ ist eine lineare Ordnung auf AZ_b .

Folgerung 2.5.20

$$i) b' \in EW_b \Rightarrow b' \in BF_h.$$

$$ii) b' \in EW_b \wedge b \text{ ist einfach} \Rightarrow b' \text{ ist wesentlich.}$$

$$iii) \text{ Sei } b_0 = \{h\}. \text{ Seien } b_1 \in EW_{b_0}, b_2 \in EW_{b_1}, \dots, b_k \in EW_{b_{k-1}} \text{ für } \forall k \in \mathbb{N}.$$

Dann sind $b_1, b_2, \dots, b_k$ wesentliche Blöcke von BF_h , für die gilt:

$$0 < w(b_1) < w(b_2) < \dots < w(b_k).$$

Bemerkung 2.5.21

Bezüglich der Aussage ii) der Folgerung 2.5.20 kann man leicht ein Beispiel (Bild 2.5.3) konstruieren, das zeigt, daß die Aussage

$$b' \text{ ist wesentlich} \Rightarrow b' \in EW_b \wedge b \text{ ist einfach}$$

ungültig ist.

In Bild 2.5.3 ist b' wesentlich, b jedoch nicht einfach.

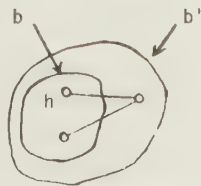


Bild 2.5.3

Satz 2.5.22

Seien $i, j, h \in V$. Seien b, b', b'' einfache Blöcke in BF_h . Seien $b' \cap b'' = b$, $b' = b \cup \{i\}$ und $b'' = b \cup \{j\}$. Seien $AZ_{b'} \neq \emptyset$, $AZ_{b''} \neq \emptyset$ die Adjazenzmengen von b' und b'' . Seien $EW_{b'}$, $EW_{b''}$ die Erweiterungsmengen von b' und b'' .

$$i \in AZ_{b''} \wedge j \in AZ_{b'} \iff \exists e (e \in EW_{b'} \cap EW_{b''} \wedge e = b \cup \{i\} \cup \{j\}).$$

Beweis:

$$\begin{aligned} " \Rightarrow " & \quad i \in AZ_{b''} \wedge j \in AZ_{b'} \\ & \Rightarrow \exists e (e \in EW_{b'} \wedge e \in EW_{b''} \wedge e = b \cup \{i\} \cup \{j\}) \\ & \Rightarrow e (e \in EW_{b'} \cap EW_{b''} \wedge e = b \cup \{i\} \cup \{j\}). \end{aligned}$$

$$" \Leftarrow " \quad \text{Angenommen: } i \notin AZ_{b''} \vee j \notin AZ_{b'}.$$

Fall 1: $i \notin AZ_{b''}$

$$\begin{aligned} & \Rightarrow b'' \cup \{i\} \notin EW_{b''} \\ & \Rightarrow e \notin EW_{b'} \cap EW_{b''}. \text{ Widerspruch.} \end{aligned}$$

Fall 2: $j \notin AZ_{b'}$

$$\begin{aligned} & \Rightarrow b' \cup \{j\} \notin EW_{b'} \\ & \Rightarrow e \notin EW_{b'} \cap EW_{b''}. \text{ Widerspruch.} \end{aligned}$$

Bemerkung 2.5.23

Bild 2.5.4 zeigt ein Beispiel für den Fall, bei dem e sowohl eine Erweiterung von b' als auch eine Erweiterung von b'' sein kann. Satz 2.5.22 legt fest, unter welcher Bedingung man von zwei voneinander unterscheidbaren Blöcken (z.B. b' und b'') ausgehend dieselbe Erweiterung (z.B. e) erhält. Bei dem noch anzugebenden Erweiterungsprozeß wird die Aussage des Satzes 2.5.22 dazu benutzt, die Konstruktion von Erweiterungen zu vermeiden, die identisch mit den bereits zustande gekommenen Erweiterungen (Blöcken) sind.

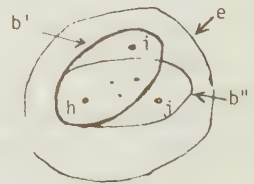


Bild 2.5.4

Bezeichnung 2.5.24

Sei $X = (x_1, \dots, x_k)$ eine Folge, für die gilt: $x_i \in V$ für $\forall i \in \mathbb{N}_k$ und $\forall i, j \in \mathbb{N}_k: (i \neq j \Rightarrow x_i \neq x_j)$. Mit $b(X)$ wird die Menge $\{x_1, \dots, x_k\}$ bezeichnet. Sei $d \in V$ und es gelte $d \neq x_i$ für $\forall i \in \mathbb{N}_k$. $X \cdot d$ bezeichnet die Folge $(x_1, \dots, x_k, d)$. Mit $\emptyset$ wird die leere Folge $()$ bezeichnet. $X = \emptyset$ genau dann, wenn $b(X) = \emptyset$. Mit $|X|$ wird die Anzahl der Komponenten in X bezeichnet. D.h. $|X| = |b(X)|$.

Definition 2.5.25

Ein Wurzelbaum $EB = (EP, EA)$ heißt genau dann Erweiterungsbaum von BF_h , wenn gilt:

- i) $EP \cong \{X \mid X = (x_1, \dots, x_k) \text{ für } \forall k \in \mathbb{N}_n \wedge x_1 = h \wedge b(X) \in BF_h\}$.
- ii) (h) ist die Wurzel von EB .
- iii) $EA \subseteq EP \times EP$, angegeben durch:

$$(X, Y) \in EA \iff \exists d (Y = X \cdot d \wedge d \in AZ_{b(X)}).$$

Folgerung 2.5.26

Sei $EB = (EP, EA)$ der Erweiterungsbaum von BF_h .

- i) $\forall X (X \in EP \iff b(X) \in BF_h \text{ ist einfach})$.
- ii) $\forall X, Y \in EP: (X = Y \Rightarrow b(X) = b(Y))$.
- iii) $EW_{b(X)} = \{b(Y) \mid (X, Y) \in EA\}$.
- iv) $\forall X, Y \in EP: (b(X) \neq b(Y) \Rightarrow X \neq Y)$.
- v) $\forall X, Y \in EP: (|X| \neq |Y| \Rightarrow b(X) \neq b(Y))$.
- vi) $\forall X \forall Y (\exists Z ((Z, X) \in EA \wedge (Z, Y) \in EA) \Rightarrow |b(X)| = |b(Y)|)$.

Definition 2.5.27

Sei $EB = (EP, EA)$ der Erweiterungsbaum von BF_h . Sei $X \in EP$.

Sei $AS_{b(X)} = (AZ_{b(X)}, VOR)$ die Adjazenzstruktur von $b(X) \in BF_h$.

Eine Erweiterungsstruktur von BF_h ist ein Quadrupel $EST \equiv (EB, ASM, p, w)$, wobei folgendes gilt:

- i) $ASM \equiv \{AS_{b(X)} \mid X \in EP\}$.

ii) $p : EP \rightarrow ASM$, angegeben durch:

$$p(X) \equiv \begin{cases} AS_{b(X)}, & \text{wenn } AZ_{b(X)} \neq \emptyset. \\ \underline{\emptyset}, & \text{wenn } AZ_{b(X)} = \emptyset. \end{cases}$$

iii) $w : EP \rightarrow \mathbb{N}$, angegeben durch:

$$w(X) = w(b(X)),$$

wobei $w(b(X))$ der Wert des Blocks $b(X) \in BF_h$ sein soll.

Folgerung 2.5.28

Sei $EST = (EB, ASM, p, w)$ die Erweiterungsstruktur von BF_h . Sei $EB = (EP, EA)$. Sei $X \in EP$.

$$i) p(X) = \underline{\emptyset} \iff EW_{b(X)} = \emptyset.$$

$$ii) p((h)) = \underline{\emptyset} \iff EB = (\{(h)\}, \emptyset).$$

$$iii) X \text{ ist ein Blatt von } EB \iff p(X) = \underline{\emptyset}.$$

Bezeichnung 2.5.29

Sei $U \subseteq V$ eine nicht leere Teilmenge von V . Man betrachtet (U, VOR) als einen gerichteten Graphen und bezeichnet die Folge $c = (u_1, --, u_1)$ als eine lineare Liste, wenn c eine Bahn (Def. B.6) in (U, VOR) ist. Mit $\text{Min } c$ wird die erste Komponente u_1 von c und mit $\text{Max } c$ die letzte Komponente u_1 von c bezeichnet. $c - \text{Min } c$ (bzw. $c - \text{Max } c$) bezeichnet die lineare Liste $(u_2, --, u_1)$ (bzw. $(u_1, --, u_{l-1})$). Sei $c_1 = (u_1, --, u_k)$, $k \leq l$, eine lineare Liste, dann bezeichnet $c_2 = c - c_1$ die lineare Liste $(u_{k+1}, --, u_l)$. Sei $d \in V$ und es gelte $d \neq u_q$ für $\forall q \in \mathbb{N}_1$, dann bezeichnet $c \cdot d$ die lineare Liste $(u_1, --, u_l, d)$, wenn d der Relation $u_l \text{ VOR } d$ genügt. Mit $\underline{\emptyset}$ wird die leere Liste $()$ bezeichnet.

Definition 2.5.30

Sei $EST = (EB, ASM, p, w)$ die Erweiterungsstruktur von BF_h . Sei $EB = (EP, EA)$. Sei $EM = \{c \mid c = (U, VOR) \wedge U \subseteq V\}$. $ZM \equiv EM \times EM$ heißt Markenmenge von EST . Ein Element $(c_1, c_2) \in ZM$ heißt Marke.

Der Erweiterungsprozeß auf EST ist ein Sechstupel

$$EWP \equiv (EST, ZM, X_0, \varepsilon, \chi, \mu).$$

i) $X_0 = (h)$ heißt Heimatknoten von EWP. $(p(X_0), \emptyset)$ heißt Anfangsmarke.

ii) $\mathcal{E}$, $\mathcal{X}$ und μ sind Abbildungen.

$\mathcal{E}: EP \times ZM \rightarrow EP \times ZM$ heißt Erweiterungsfunktion.

$\mathcal{X}: EP \times ZM \rightarrow EP$ heißt Wurzelsuchfunktion.

$\mu: EP \times ZM \rightarrow ZM$ heißt Markierungsfunktion.

$\mathcal{E}$, $\mathcal{X}$ und μ werden folgendermaßen angegeben:

Fall 1: $c_1 \neq \emptyset$.

$$\mathcal{E}((X, (c_1, c_2))) = (X', (c'_1, c'_2)), \text{ wobei gilt:}$$

$$X' = X \circ \text{Min } c_1.$$

$$c'_1 = \begin{cases} p(X') - c_2, & \text{falls } p(X') \neq \emptyset. \\ \emptyset, & \text{falls } p(X') = \emptyset. \end{cases}$$

$$c'_2 = \emptyset.$$

$$\mu((X, (c_1, c_2))) = (c_1 - \text{Min } c_1, c_2 \circ \text{Min } c_1).$$

$\mathcal{X}$ ist für den Fall 1 undefiniert.

Fall 2: $c_1 = \emptyset$

$$\mathcal{X}((X, (c_1, c_2))) = \begin{cases} X - \text{Max } X, & \text{falls } X \neq X_0. \\ X, & \text{falls } X = X_0. \end{cases}$$

$$\mu((X, (c_1, c_2))) = (c_1, c_2).$$

$\mathcal{E}$ ist für den Fall 2 undefiniert.

Sei $(X, (c_1, c_2)) \in EP \times ZM$. Dann heißt (c_1, c_2) die Marke von X. Anders ausgedrückt, X ist mit (c_1, c_2) markiert.

Folgerung 2.5.31

Seien $(X, (c_1, c_2)), (X', (c'_1, c'_2)) \in EP \times ZM$.

$$i) \mathcal{E}((X, (c_1, c_2))) = (X', (c'_1, c'_2))$$

$$\Rightarrow (X, X') \in EA \wedge b(X') \in EW_{b(X)}.$$

$$\text{ii) } \chi((X, (c_1, c_2))) = X'' \wedge X \neq X_0 \\ \Rightarrow (X'', X) \in EA \wedge b(X) \in EW_b(X'')$$

Bemerkung 2.5.32

Zwischen dem eben definierten Erweiterungsprozeß und dem in /16/ behandelten "Erweiterungsprinzip" besteht überhaupt kein Zusammenhang. Das Erweiterungsprinzip ist im Grunde eine Verallgemeinerung des Verzweigungsprinzips ("branch-and-bound"). Dagegen wird der Erweiterungsprozeß als ein Rechenprozeß mit spezieller Aufgabe, nämlich der Auffindung aller einfachen Blöcke einer Blockfamilie, betrachtet.

Das Verhalten des Erweiterungsprozesses läßt sich ähnlich wie das Verhalten des binären Verzweigungsprozesses (Def. 2.4.27) beschreiben.

Definition 2.5.33

Sei $EST = ((EP, EA), ASM, p, w)$ eine Erweiterungsstruktur von BF_h . Sei $EWP = (EST, ZM, X_0, \mathcal{E}, \chi, \mu)$ der Erweiterungsprozeß auf EST . Seien $\mathcal{E}((X, (c_1, c_2))) = (X', (c'_1, c'_2))$, $\chi((X, (c_1, c_2))) = X''$ und $\mu((X, (c_1, c_2))) = (c''_1, c''_2)$.

EWP ist im Zustand laufbereit, wenn $c_1 = \emptyset$ und $X = X_0$. In allen anderen Fällen, in denen $\mathcal{E}, \chi$ und μ gemäß der Definition 2.5.30 angegeben werden, ist EWP im Zustand laufend. Ein EWP, der sich im Zustand "laufend" ("laufbereit") befindet, heißt laufender (laufbereiter) Erweiterungsprozeß.

Jeder Knoten $X \in EP$ besitzt 3 Zustände: wartend, aktiv und inaktiv. X ist g.d. (im Zustand) wartend, wenn X nicht markiert ist. Sei X mit (c_1, c_2) markiert. X ist g.d. aktiv, wenn $c_1 \neq \emptyset$. Gilt $c_1 = \emptyset$, dann ist X inaktiv.

Ein Übergang von Knoten X zu Knoten X' (bzw. X zu X'') ist genau dann möglich, wenn $c_1 \neq \emptyset$ und $X' = X \circ \text{Min } c_1$ (bzw. wenn $c_1 = \emptyset$ und $X'' = X - \text{Max } X$). X heißt Ausgangsknoten. X' und X'' heißen Zielknoten. Der EWP markiert den Knoten X , wenn X der Ausgangsknoten von EWP ist. Jeder Übergang wird als ein Schritt des EWP bezeichnet.

Der EWP geht einen Gang durch den Erweiterungsbaum $EB = (EP, EA)$, wenn er, von dem Anfangsknoten $X_0 = (h)$ ausgehend, der identisch mit der Wurzel von EB ist, von einem Knoten zu einem anderen übergeht und bei jedem Schritt den Ausgangsknoten markiert. Ein Gang ist genau dann eine Tour, wenn EWP einen Schritt zu X_0 macht und sich dort in den Zustand "laufbereit" versetzt.

Das Zeitintervall, in dem eine Tour des EWP stattfindet, heißt Laufphase von EWP. Die Anzahl der Schritte, die der EWP bei einer Tour machen muß, heißt Aufwand des EWP. Der Aufwand wird mit $AW(EWP)$ bezeichnet.

Bemerkung 2.5.34

Die Arbeitsweise eines EWP kann man sich folgendermaßen vorstellen: Sei X der Ausgangsknoten eines laufenden EWP. Sei (c_1, c_2) die Marke von X . Ist X aktiv (d.h. $c_1 \neq 0$), so geht EWP zu X' über, wobei $(X, X') \in EA$ und X' und (c'_1, c'_2) gemäß der Erweiterungsfunktion angegeben sind. Bei diesem Übergang ändert sich die Marke von X gemäß der Markierungsfunktion. Der Übergang wird als "Vorwärtsschritt" bezeichnet. Befindet sich X im Zustand inaktiv und es gilt $X \neq X_0$, so geht EWP zu X'' über, wobei $X'' = X - \max X$. In diesem Falle wird der Übergang als "Rückwärtsschritt" bezeichnet. Ist X inaktiv und es gilt $X = X_0$, dann beendet der EWP bei Knoten X seinen Laufzustand und versetzt sich in den Zustand "laufbereit". In den beiden letzteren Fällen verändert sich weder die Markierung von X noch die Markierung von X'' . Eine Tour des EWP durch den Erweiterungsbaum läßt sich als ein Prozeß zur Konstruktion eines Teilbaums $EB' = (EP', EA')$ betrachten, wobei $EP' \subseteq EP$, angegeben durch $EP' = \{X \mid X \in EP \wedge X \text{ ist markiert}\}$, und $EA' = (EP' \times EP') \cap EA$. Bei einem Vorwärtsschritt geht EWP stets von einem Knoten X zu einem Knoten X' über, wobei gilt: $(X, X') \in EA$ und $b(X') \in EW_{b(X)}$ (Fol. 2.5.31i). Da $b(X')$ eine Erweiterung von $b(X)$ sein muß, wird der Prozeß Erweiterungsprozeß genannt. Da mehrere Erweiterungen von $b(X)$ existieren können, besteht die Aufgabe des Erweiterungsprozesses darin, von einem gegebenen Knoten X ausgehend, ausschließlich diejenigen Erweiterungen von $b(X)$ aufzufinden, die nicht durch die bereits inaktiven Knoten der Erweiterungsstruktur angegeben sind (Satz 2.5.22 und Bemerkung 2.5.23).

Beispiel 2.5.35

Ein Netz $NG = (V, E, g, f)$ (Bild 2.5.5) wird folgendermaßen angegeben:

$$V = \{1, 2, 3, 4, 5\}.$$

$$E = \{\{1,2\}, \{1,4\}, \{2,3\}, \{3,4\}, \{4,5\}\}.$$

$$g = (1, 1, 1, 1, 1).$$

$$f = \begin{pmatrix} 0 & 1 & 0 & 6 & 0 \\ 1 & 0 & 10 & 0 & 0 \\ 0 & 10 & 0 & 5 & 0 \\ 6 & 0 & 5 & 0 & 2 \\ 0 & 0 & 0 & 2 & 0 \end{pmatrix}$$

$a = 3$ ist die Gewichtsgrenze.

$(V, \text{VOR}) = (1, 2, 3, 4, 5)$.

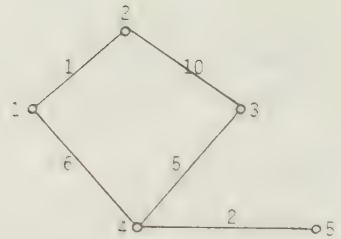


Bild 2.5.5

Bild 2.5.6 zeigt den Erweiterungsbaum $EB = (EP, EA)$ von BF_1 , wobei

$EP = \{(1), (1,2), (1,4), (1,2,3), (1,2,4), (1,4,2), (1,4,3), (1,2,5)\}$.

$EA = \{((1), (1,2)), ((1), (1,4)), ((1,2), (1,2,3)), ((1,2), (1,2,4)), ((1,4), (1,4,2)), ((1,4), (1,4,3)), ((1,4), (1,4,5))\}$.

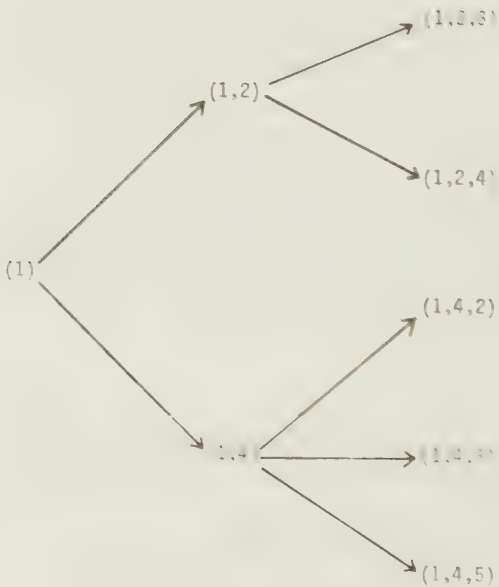


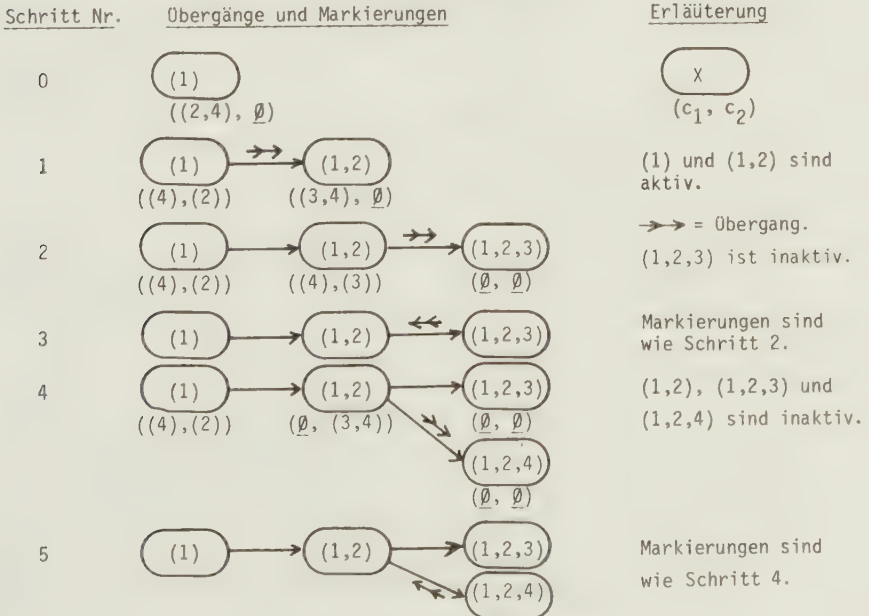
Bild 2.5.6

Die verschiedenen Größen der Erweiterungsstruktur $EST = ((EP, EA), ASM, p, w)$ werden in der folgenden Tabelle angegeben:

$X \in EP$	$AZ_b(X)$	$p(X) \in ASM$	$w(X)$
(1)	{2,4}	(2,4)	0
(1,2)	{4,3}	(3,4)	1
(1,4)	{3,5,2}	(2,3,5)	6
(1,2,3)	$\emptyset$	$\underline{\emptyset}$	11
(1,2,4)	$\emptyset$	$\underline{\emptyset}$	7
(1,4,2)	$\emptyset$	$\underline{\emptyset}$	7
(1,4,3)	$\emptyset$	$\underline{\emptyset}$	11
(1,4,5)	$\emptyset$	$\underline{\emptyset}$	8

In Bild 2.5.7 werden die einzelnen Schritte des EWP und die Markierungen der einzelnen Knoten von EST angegeben. Dieses Bild zeigt, wie der Teilbaum von EB, $EB' = (EP', EA')$ (Bemerkung 2.5.34) bei der Tour des Erweiterungsprozesses Schritt für Schritt konstruiert wird. In diesem Beispiel gilt $(1,4,2) \notin EP'$, d.h. (1,4,2) ist der einzige Knoten von EST, der sich nach einer Tour des EWP im Zustand "wartend" befindet.

Bild 2.5.7



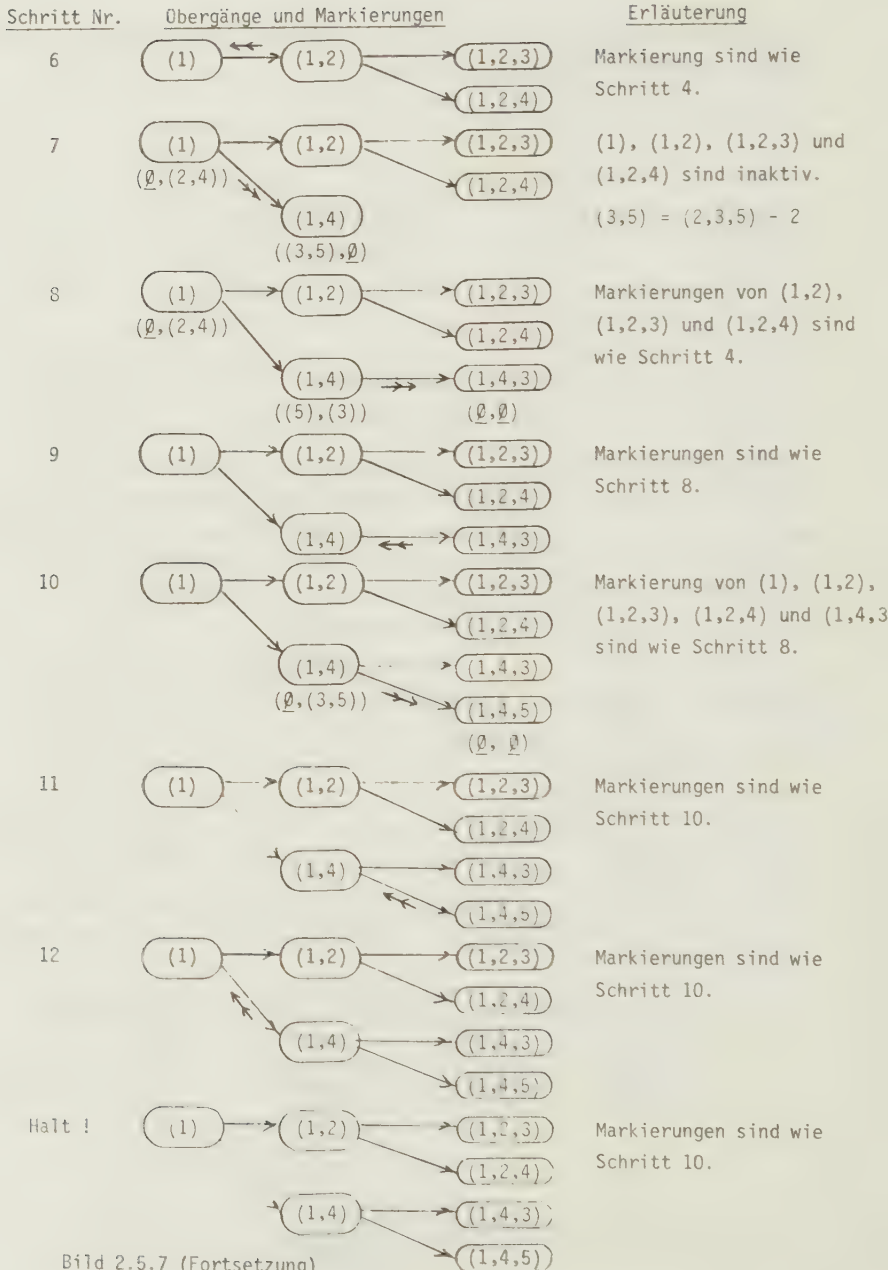


Bild 2.5.7 (Fortsetzung)

Im folgenden sei $EST = (EB, ASM, p, w)$ die Erweiterungsstruktur von BF_h . Sei $EB = (EP, EA)$. Sei $EWP = (EST, EM, X_0, \xi, \chi, \mu)$ der Erweiterungsprozeß auf EST .

Satz 2.5.36

Sei $EB' = (EP', EA')$ ein Teilbaum von EB , für den gilt:

$$EP' = \{X \mid X \in EP \wedge X \text{ ist markiert}\}, \quad EA' = (EP' \times EP') \cap EA.$$

- i) EB' ist ein Wurzelbaum mit der Wurzel (h) .
- ii) Sei $(X, X') \in EA'$. EWP macht genau zwei Übergänge zwischen X und X' .
- iii) $AW(EWP) = 2 \mid EA' \mid = 2(\mid EP' \mid - 1)$.
- iv) $\forall X(X \in EP' \iff b(X) \in BF_h \text{ ist einfach})$.

Beweis:

- i) Nach Def. 2.5.25 ist EB ein Wurzelbaum mit der Wurzel (h) . $EB' = (EP', EA')$ ist ein Teilbaum von EB . EP' enthält alle Knoten von EB , die bei einer Tour markiert sind. (h) bildet die Wurzel von EB' , da (h) der Heimatknoten von EWP ist. Nach Def. 2.5.33 und Bemerkung 2.5.34 ist ein Übergang zwischen zwei Knoten X, X' genau dann möglich, wenn $(X, X') \in EA$ oder $(X', X) \in EA$. Dies bedeutet, daß EB' zusammenhängend sein muß. EB' ist also ein Wurzelbaum mit der Wurzel (h) .
- ii) Sei (c_1, c_2) die Marke von X . Sei (c'_1, c'_2) die Marke von X' .
Nach Def. 2.5.30 macht EWP von X aus einen Übergang zu X' , wenn $c_1 \neq \emptyset$ und $X' = X \cdot \text{Min } c_1$. Dabei erhält X die neue Marke $(c_1 - \text{Min } c_1, c_2 \cdot \text{Min } c_1)$. Von X' aus kann EWP so lange Vorwärtsschritte machen, bis $c'_1 = \emptyset$ wird. Nur in diesem Falle macht EWP einen Rückwärtsschritt von X' nach X . Von X aus kann EWP wieder Vorwärtsschritte machen, so lange $c_1 \neq \emptyset$ gilt. Da c_1 sich bei jedem Vorwärtsschritt, der von X aus gemacht wird, ändert, führt jeder von diesen Vorwärtsschritten zu einem anderen Knoten, der $\neq X'$ ist. Zwischen X und X' können deshalb genau zwei Schritte gemacht werden: ein Vorwärtsschritt und ein Rückwärtsschritt.
- iii) $AW(EWP) = 2 \mid EA' \mid$ ergibt sich aus der Aussage ii), da bei jeder Kante von EB' genau zwei Übergänge bei einer Tour stattfinden können. Für den Baum EB' gilt $\mid EA' \mid = \mid EP' \mid - 1$ (Satz B.12). Daraus folgt $AW(EWP) = 2(\mid EP' \mid - 1)$.
- iv) Da $EP' \subseteq EP$, so ergibt sich diese Aussage aus der Folgerung 2.5.26 i. $\square$

Bemerkung 2.5.37

Eine unmittelbare Folgerung der Aussage iii) des Satzes 2.5.36 besteht darin, daß statt $AW(EWP)$ die Anzahl der bei einer Tour markierten Knoten ($|EP'|$) als Aufwand des EWP betrachtet werden kann. Dies hat den Vorteil, daß man die Anzahl der Schritte während der Tour des EWP nicht aufzuzählen braucht. Wir werden künftig $EB' = (EP', EA')$ als markierten Erweiterungsbaum bezeichnen.

Lemma 2.5.38

$$\forall X, Y \in EP: (X \neq Y \wedge \exists Z((Z, X) \in EP \wedge (Z, Y) \in EP) \Rightarrow |X| = |Y| \wedge b(X) \neq b(Y)).$$

Beweis:

X und Y haben einen gemeinsamen Vater Z . Es gilt also nach Def. 2.5.25 $|X| = |Y|$.

Aus $X \neq Y$ folgt, daß es $d, d' \in V, d \neq d'$, gibt, sodaß

$$X = Z \circ d \quad \text{und} \quad Y = Z \circ d'.$$

Dies bedeutet, daß $b(X) = b(Z) \cup \{d\}$ und $b(Y) = b(Z) \cup \{d'\}$.

Daraus folgt, daß $b(X) \neq b(Y)$. $\square$

Lemma 2.5.39

Seien $(S, X), (T, Y) \in EA$ und es gelte $X \neq Y \wedge |X| = |Y|$.

$$\exists e(b(S) \cap b(T) = e \wedge |b(S) \setminus e| = |b(T) \setminus e| \geq 2) \Rightarrow b(X) \neq b(Y).$$

Beweis:

$$\exists e(b(S) \cap b(T) = e \wedge |b(S) \setminus e| = |b(T) \setminus e| \geq 2)$$

$$\Rightarrow \exists e \exists b' \exists b'' (b(S) = e \cup b' \wedge b(T) = e \cup b'' \wedge |b'| = |b''| \geq 2 \wedge$$

$$\forall l, k (l \in b' \wedge k \in b'' \Rightarrow l \neq k))$$

$$\Rightarrow \exists e \exists b' \exists b'' \exists d' \exists d'' (b(X) = e \cup b' \cup \{d'\} \wedge b(Y) = e \cup b'' \cup \{d''\} \wedge |b'| = |b''| \geq 2 \wedge$$

$$\forall l, k (l \in b' \wedge k \in b'' \Rightarrow l \neq k))$$

$$\Rightarrow b(X) \neq b(Y). \quad \square$$

Lemma 2.5.40

Seien $(S, X), (T, Y) \in EA'$ und es gelte $(X \neq Y \wedge |X| = |Y|)$.

$$\exists e(b(S) \cap b(T) = e \wedge |b(S) \setminus e| = |b(T) \setminus e| = 1) \Rightarrow b(X) \neq b(Y).$$

Beweis:

Sei (c_1, c_2) die Marke von S. Sei (c'_1, c'_2) die Marke von T.

Seien $b(S) = e \cup \{d\}$, $b(T) = e \cup \{d'\}$, wobei $d \neq d'$.

Angenommen S wird markiert bevor T markiert wird. Für X, den Sohn von S, gilt folgendes:

$$X = S \bullet \text{Min } c_1 \text{ und } b(X) = b(S) \cup \{\text{Min } c_1\} = e \cup \{d\} \cup \{\text{Min } c_1\}.$$

Um T zu markieren, muß der EWP von S aus mindestens einen Rückwärtsschritt machen, bis er einen gemeinsamen echten Vorgänger von S und T, z.B. Q, erreicht hat. Von Q aus muß der EWP mindestens einen Vorwärtsschritt machen, um T zu markieren. Hierbei lassen sich die folgenden zwei Fälle unterscheiden:

Fall 1: Q ist ein gemeinsamer Vater von S und T.

In diesem Falle gilt: $S = Q \bullet d$ und $T = Q \bullet d'$.

Da S vor T markiert wird, so folgt aus Def. 2.5.30, daß d nicht eine Komponente von c'_1 sein kann. D.h. $d \neq \text{Min } c'_1$.

Aus $b(X) = e \cup \{d\} \cup \{\text{Min } c_1\}$, $b(Y) = e \cup \{d'\} \cup \{\text{Min } c'_1\}$ und $d \neq \text{Min } c'_1$ folgt $b(X) \neq b(Y)$.

Fall 2: Q ist ein gemeinsamer, echter Vorgänger, jedoch nicht der Vater von S und T.

Seien R, R' die Söhne von Q. Sei R der echte Vorgänger von S und R' der echte Vorgänger von T. Dies bedeutet, daß $R = Q \bullet d$ und $R' = Q \bullet d'$, wobei $d \neq d'$. Sei (c''_1, c''_2) die Marke von R'. Dann folgt aus Def. 2.5.30, daß d nicht eine Komponente von c''_1 sein kann. D.h. das Element $d \in V$ gehört nicht zu denjenigen Komponenten, die die Marken von $R'_1, R'_2, \dots, T$ bilden, wobei $R'_1, R'_2, \dots, T$ die Nachfolger von R' sein sollen. Dies ist gleichbedeutend mit $d \neq \text{Min } c''_1$. Daraus folgt $b(X) \neq b(Y)$.

Die gleiche Beweisführung gilt, wenn T vor S markiert wird. $\square$

Satz 2.5.41

$$\forall X, Y \in EP': (X \neq Y \iff b(X) \neq b(Y)).$$

Beweis:

" $\implies$ " Für $X \neq Y$ lassen sich die folgenden zwei Fälle voneinander unterscheiden:

$$(1) |X| \neq |Y|$$

$$(2) |X| = |Y|.$$

Fall 1: $|X| \neq |Y|$.

Nach Aussage v) der Folgerung 2.5.26 gilt in diesem Falle $b(X) \neq b(Y)$.

Fall 2: $|X| = |Y| = k$.

Wir betrachten ausschließlich den Fall $k \geq 2$, da aus $k = 1$ $X = Y = (h)$ folgen muß.

Vollständige Induktion über k für $k \geq 2$:

$k = 2$: (h) ist der gemeinsame Vater von X und Y . Aus Lemma 2.5.38 folgt

$$X \neq Y \implies b(X) \neq b(Y).$$

Induktionsvoraussetzung: $X \neq Y \implies b(X) \neq b(Y)$ für $k = 1$.

Sei $k = 1 + 1$. Seien $(S, X), (T, Y) \in EA'$.

Wenn $S = T$, dann haben X und Y einen gemeinsamen Vater. Es gilt nach Lemma 2.5.38:

$$X \neq Y \implies b(X) \neq b(Y).$$

Wenn $S \neq T$, dann folgt aus der Induktionsvoraussetzung $b(S) \neq b(T)$. Für $b(S)$ und $b(T)$ bestehen die folgenden zwei Möglichkeiten:

$$i) \exists e (b(S) \cap b(T) = e \wedge |b(S) \setminus e| = |b(T) \setminus e| \geq 2) \implies b(X) \neq b(Y).$$

(Lemma 2.5.39)

$$ii) \exists e' (b(S) \cap b(T) = e' \wedge |b(S) \setminus e'| = |b(T) \setminus e'| = 1) \implies b(X) \neq b(Y).$$

(Lemma 2.5.40)

" $\impliedby$ "

Diese Aussage ist identisch mit der Aussage iV der Folgerung 2.5.26. $\square$

Korollar 2.5.42

Sei $EB' = (EP', EA')$ der markierte Erweiterungsbaum von BF_h .

$$EP' = \{x \mid b(x) \in BF_h \wedge b(x) \text{ ist einfach}\}.$$

Beweis:

Diese Aussage ergibt sich aus Satz 2.5.41 und der Aussage i) der Folgerung 2.5.26. $\square$

Bemerkung 2.5.43

Die einfachen Blöcke einer Blockfamilie werden also durch die Knoten des entsprechenden, markierten Erweiterungsbaums angegeben. Man kann also den Erweiterungsprozeß als einen Rechenprozeß betrachten, der die Menge aller einfachen Blöcke einer Blockfamilie auffindet. Sei BE_h die Menge aller einfachen Blöcke von BF_h , dann ist $BW_h = BE_h \setminus \{h\}$ die Menge aller wesentlichen Blöcke von BF_h . Aus BW_h läßt sich dann K_h , der Kern der BIR-Familie F_h , ableiten.

Beispiel 2.5.44

Die markierten Erweiterungsbäume aller Blockfamilien des Netzes des Beispiels 2.3.45 werden in Bild 2.5.9 angegeben. Hierbei scheint vorteilhaft zu sein, die Zeilen und die Spalten der Matrix f zunächst gemäß der Knotenrelation VOR umzuordnen (Bild 2.5.8).

	11	1	3	7	5	12	4	9	8	6	2	10
11	0	0	0	0	10	1	0	0	0	0	0	11
1	0	0	0	0	0	0	0	0	0	0	1	0
3	0	0	0	5	0	0	45	0	0	0	50	0
7	0	0	5	0	0	0	0	0	21	25	9	0
5	10	0	0	0	0	0	5	0	10	35	0	0
12	1	0	0	0	0	0	0	0	0	0	0	0
4	0	0	45	0	5	0	0	0	0	0	40	0
9	0	0	0	0	0	0	0	0	11	0	0	69
8	0	0	0	21	10	0	0	11	0	0	0	0
6	0	0	0	25	35	0	0	0	0	0	0	0
2	0	1	50	9	0	0	40	0	0	0	0	0
10	11	0	0	0	0	0	0	69	0	0	0	0

Bild 2.5.8

$h \in V$	$EB' = (EP', EA')$	$\{b(X) \mid X \in EP' \setminus \{(h)\}\}$	$w(b(X))$
11	(11)	$\emptyset$	
1	(1) $\longrightarrow$ (1,2)	$\{\{1,2\}\}$	1
3	(3) $\longrightarrow$ (3,2)	$\{\{3,2\}\}$	50
7	(7) $\longrightarrow$ (7,6) $\searrow$ (7,2)	$\{\{7,6\}, \{7,2\}\}$	25 9
5	(5) $\longrightarrow$ (5,8) $\searrow$ (5,6)	$\{\{5,8\}, \{5,6\}\}$	10 35
12	(12)	$\emptyset$	
4	(4) $\longrightarrow$ (4,2)	$\{\{4,2\}\}$	40
9	(9) $\longrightarrow$ (9,8) $\longrightarrow$ (9,8,10) $\searrow$ (9,10)	$\{\{9,8\}, \{9,8,10\}, \{9,10\}\}$	11, 80 69
8	(8)	$\emptyset$	
6	(6)	$\emptyset$	
2	(2)	$\emptyset$	
10	(10)	$\emptyset$	

Bild 2.5.9

Bemerkung 2.5.45

Die in Bild 2.5.1 (Beispiel 2.5.17) angegebene K-Liste lässt sich sofort aus der dritten und der vierten Spalte des Bilds 2.5.9 ableiten.

Satz 2.5.46

Sei $EB'_h = (EP'_h, EA'_h)$ der markierte Erweiterungsbaum von BF_h . Sei WB die Menge aller wesentlichen Blöcke des Netzes $NG = (V, E, g, f)$. Sei β die relative Blockkapazität des Netzes NG . Sei $k = \lceil \beta \rceil$.

$$i) \quad WB = \{b(X) \mid X \in \bigcup_{h \in V} (EP'_h \setminus \{(h)\})\}.$$

$$ii) \quad 0 \leq |WB| \leq \sum_{i=2}^k \binom{n}{i}.$$

Beweis:

- i) Nach Korollar 2.5.42 enthält EP'_h ausschließlich Knoten, die den einfachen Blöcken von BF_h entsprechen. Davon ist (h) der einzige Knoten, der den redundanten Block $\{h\}$ angibt. Die übrigen einfachen Blöcke von BF_h sind wesentlich.
- ii) $|WB| \geq 0$ gilt offensichtlich.

Der obere Grenzwert von $|WB|$ hängt von zwei Parametern ab: die relative Blockkapazität β und die Struktur des Graphen (V, E) .

Bei einer gegebenen Blockkapazität β wird $|WB|$ maximal, wenn (V, E) ein vollständiger Graph ist. Da in diesem Falle $|AZ_{b(X)}|$ für $\forall x \in EP'_h$ maximal sein muß (Def. 2.5.18).

Bei einem gegebenen vollständigen Graphen nimmt die Anzahl der einfachen Blöcke in BF_h mit β zu. $\lceil \beta \rceil$ läßt sich als Gewichtsgrenze in dem Falle betrachten, in dem $g_1 = \dots = g_n = 1$ gilt. Der obere Grenzwert von $|WB|$ ergibt sich, wenn (V, E) einen vollständigen Graphen darstellt, dessen Knotengewichte Eins sind und bei dem $k = \lceil \beta \rceil$ die Gewichtsgrenze angibt. In diesem Falle gilt:

$$WB_0 = \bigcup_{i=2}^k \binom{V}{i}.$$

$|WB_0| = \sum_{i=2}^k \binom{n}{i}$ ist also der obere Grenzwert von $|WB|$. $\square$

Bemerkung 2.5.47

$|WB|$ läßt sich als Gesamtaufwand der Erweiterungsprozesse in der Phase 1 betrachten.

Bemerkung 2.5.48

Wenn (V, E) ein vollständiger Graph ist und sich k dem Wert $n = |V|$ nähert, dann nimmt $|WB|$ proportional mit $2^n = \sum_{i=0}^n \binom{n}{i}$ zu. Da $|WB|$ die Anzahl der in der K -Liste enthaltenen Elemente angibt, kann dies bei großem n sehr großen Aufwand des binären Verzweigungsprozesses verursachen. Theoretisch kann man stets ein Netz konstruieren, dessen Struktur einen vollständigen Graphen darstellt und dessen relative Blockkapazität sich der Mächtigkeit der Knotenmenge nähert. Die in der Praxis auftretenden Probleme bestehen jedoch aus Netzen, deren Strukturen durch unvollständige Graphen angegeben sind und bei denen die Elemente der Matrix f zum größten Teil aus Nullen bestehen. Dies gilt insbesondere dann, wenn das zu zerlegende Netz groß ist.

2.6 Der Ergänzungsprozeß

Die sich nach einer Tour des binären Verzweigungsprozesses ergebende Lösung des Problems P2 entspricht einer Menge von Blöcken, deren Gesamtwert das Optimum des Problems $P = (NG, a, Z)$ angibt. Wie bereits in Abschnitt 2.2 erwähnt, stellt diese Menge von Blöcken in den meisten Fällen leider nicht die Zerlegung des Netzes dar, da sie der Bedingung $\bigcup_{i \in N_m} b_i = V$ (Def. 2.1.6)

nicht genügt. Der Rechenprozeß der Phase 3 muß eingeleitet

werden, damit man eine optimale Zerlegung mit möglichst gutem Ausdehnungsgrad erhält. Die derart "vervollständigte" Lösung des Problems P2 bildet dann die endgültige Lösung des Problems P. Der Rechenprozeß der Phase 3, Ergänzungsprozeß genannt, und die damit zusammenhängenden Strategien werden in diesem Abschnitt entwickelt und untersucht.

In diesem Abschnitt sei $NG = (V, E, g, f)$ ein ungerichtetes Netz. Sei a die Gewichtsgrenze. Sei $VST = (P, A, KF, \rho, \omega, \varphi, \hat{Z})$ eine binäre Verzweigungsstruktur bezüglich der K-Liste von NG. Sei $|KF| = 1$. Sei $VP = (VST, MARZ, 0)$ der binäre Verzweigungsprozeß auf VST. Sei ZB die Menge der zulässigen Blätter, MB die Menge der maximalen Blätter von VST. Sei $\hat{Z}$ das Maximum von VST. Sei WB die Menge aller wesentlichen Blöcke, EF die Menge aller einfachen Blöcke von NG.

Definition 2.6.1

Sei $\underline{u} \in ZB$ und $\underline{u} = (u_1, \dots, u_1)$.

$$BM(\underline{u}) \equiv \left\{ b_i \mid i \in I_{\underline{u}} \wedge b_i \text{ ist ein Block, dessen BIR } (\underline{s}_i, w_i) \in KF \text{ ist.} \right\}$$

heißt zugeordnete Blockmenge (kurz, Blockmenge) von $\underline{u}$. $BM(\underline{u})$ heißt optimal genau dann, wenn $\underline{u} \in MB$.

Folgerung 2.6.2

$$i) \quad b \in BM(\underline{u}) \Rightarrow b \in WB.$$

$$ii) \quad \underline{u} \in MB \Rightarrow \omega(\underline{u}) = \sum_{b \in BM(\underline{u})} w(b) = \hat{Z}.$$

$$iii) \quad \forall b, b' \in BM(\underline{u}) : (b \neq b' \Rightarrow b \cap b' = \emptyset).$$

$$iv) \quad \bigcup_{b \in BM(\underline{u})} b \subseteq V.$$

Satz 2.6.3

Sei $\underline{u} \in MB$.

$$\bigcup_{b \in BM(\underline{u})} b = V \iff (BM(\underline{u}), \omega(\underline{u})) \text{ ist eine Lösung des Problems P.}$$

Beweis:

" $\Rightarrow$ " $\underline{u} \in MB \subseteq ZB$. $BM(\underline{u})$ genügt den Aussagen ii) und iii) der Folgerung 2.6.2.

Mit $\bigcup_{b \in BM(\underline{u})} b = V$ genügt $(BM(\underline{u}), \omega(\underline{u}))$ allen Bedingungen der Definition

2.1.6. $(BM(\underline{u}), \omega(\underline{u}))$ ist also eine Lösung des Problems P.

" $\Leftarrow$ " Offensichtlich. $\square$

Bemerkung 2.6.4

Der Fall $\underline{u} \in MB$ und $\bigcup_{b \in BM(\underline{u})} b \subsetneq V$ kommt sehr oft bei den in der Praxis auftretenden Problemen vor. Dabei wird $BM(\underline{u})$ als die Teilmenge einer optimalen Zerlegung betrachtet und man interessiert sich an erster Stelle für die Frage, wie man aus $BM(\underline{u})$ eine optimale Zerlegung erhält, deren Ausdehnungsgrad möglichst gut sein soll.

Definition 2.6.5

Die Abbildung $r : 2^V \rightarrow \mathbb{N}_0$, angegeben durch

$$r(b) = \begin{cases} a - \sum_{j \in b} g_j, & \text{wenn } a \geq \sum_{j \in b} g_j. \\ \text{undefiniert,} & \text{wenn } a < \sum_{j \in b} g_j. \end{cases}$$

heißt Restkapazität des Blocks b .

b heißt voll, wenn $r(b) = 0$. Sonst heißt b mäßig.

Bemerkung 2.6.6

Sei $\underline{u} \in ZB$. $r(b) \geq 0$ für $\forall b \in BM(\underline{u})$.

Definition 2.6.7

Sei $\underline{u} \in MB$. Sei $BM(\underline{u})$ die Blockmenge von $\underline{u}$.

$$D(\underline{u}) \cong V \setminus \bigcup_{b \in BM(\underline{u})} b$$

heißt Ergänzung von $BM(\underline{u})$.

$$\check{g} \cong \min \{g_i \mid i \in D(\underline{u})\}$$

heißt Ergänzungsgewicht von $BM(\underline{u})$.

Definition 2.6.8

Sei $\underline{u} \in MB$. Sei $BM(\underline{u})$ die Blockmenge von $\underline{u}$. $B(\underline{u})$ heißt die durch $BM(\underline{u})$ induzierte optimale Zerlegung genau dann, wenn gilt:

$$B(\underline{u}) \cong BM(\underline{u}) \cup \binom{D(\underline{u})}{1}.$$

Bezeichnung 2.6.9

Betrachtet man das Ergänzungsgewicht $\check{g}$ als Bezugsgröße, so läßt sich sowohl $D(\underline{u})$ als auch $BM(\underline{u})$ in zwei Teilmengen aufteilen, die folgendermaßen bezeichnet werden:

$$\begin{aligned} \text{Für } D(\underline{u}): \quad DV(\underline{u}) &\cong \{i \mid i \in D(\underline{u}) \wedge g_i + \check{g} > a\}. \\ DR(\underline{u}) &\cong D(\underline{u}) \setminus DV(\underline{u}). \end{aligned}$$

$$\begin{aligned} \text{Für } BM(\underline{u}): \quad BA(\underline{u}) &\cong \{b \mid b \in BM(\underline{u}) \wedge \check{g} > r(b)\}. \\ BB(\underline{u}) &\cong BM(\underline{u}) \setminus BA(\underline{u}). \end{aligned}$$

Folgerung 2.6.10

- i) $D(\underline{u}) = \emptyset \iff B(\underline{u}) = BM(\underline{u})$.
- ii) $\sum_{b \in \binom{D(\underline{u})}{1}} w(b) = 0$.
- iii) $\underline{u} \in MB \implies \sum_{b \in B(\underline{u})} w(b) = \hat{Z} = \sum_{b \in BM(\underline{u})} w(b)$.

Bemerkung 2.6.11

$(B(\underline{u}), \hat{Z})$ ist eine Lösung des Problems P. Falls $D(\underline{u}) \neq \emptyset$, dann besteht die Möglichkeit, daß mindestens ein $(B'(\underline{u}), \hat{Z})$ existiert, sodaß $|B(\underline{u})| \geq |B'(\underline{u})|$, d.h. man sollte in diesem Falle versuchen, eine optimale Zerlegung mit besserem Ausdehnungsgrad zu finden. Das Problem, den Ausdehnungsgrad einer induzierten optimalen Zerlegung zu minimieren, gehört zu einem speziellen Fall des allgemeinen Ladeproblems /74/, nämlich dem sogenannten "Bin-Packing" Problem (z.B. /79, 80/). Zur Lösung dieses Problems sind sowohl exakte Rechenverfahren als auch gute heuristische Algorithmen vorhanden. Der noch anzugebende Ergänzungsprozeß beruht im Grunde auf den Ideen der in /101/ untersuchten Heuristik.

Lemma 2.6.12

Sei $\underline{u} \in MB$ und $b \in BM(\underline{u})$. Sei $D(\underline{u}) \neq \emptyset$.

$$\forall j \in D(\underline{u}): (\exists i (i \in b \wedge (i, j) \in E) \Rightarrow g_j > r(b)).$$

Beweis:

Angenommen es gibt ein $j \in D(\underline{u})$, sodaß $g_j \leq r(b)$. D.h. $g_j + \sum_{k \in b} g_k \leq a$.

Ein neuer Block $b' = b \cup \{j\}$ kann also gebildet werden. Da $(i, j) \in E$, so ist b' wesentlich (Def. 2.5.3) und es gilt: $w(b') = w(b) + f_{ij} > w(b)$. Dies bedeutet, daß ein $\underline{u}' \in MB$ existiert, dessen Blockmenge folgendermaßen angegeben wird:

$$BM(\underline{u}') = (BM(\underline{u}) \setminus \{b\}) \cup \{b'\}.$$

Daraus folgt: $\omega(\underline{u}') = \omega(\underline{u}) - w(b) + w(b') > \omega(\underline{u}) = \hat{Z}$. Dies steht im Widerspruch zu der Voraussetzung, daß $\underline{u} \in MB$. $\square$

Satz 2.6.13

Sei $\underline{u} \in MB$.

$$\forall b \in BM(\underline{u}), \forall j \in D(\underline{u}): (g_j \leq r(b) \Rightarrow \forall i (i \in b \Rightarrow (i, j) \notin E)).$$

Beweis:

Angenommen: $\exists i (i \in b \wedge (i, j) \in E)$.

Nach Lemma 2.6.12 bedeutet dies, daß $g_j > r(b)$. Widerspruch. $\square$

Bemerkung 2.6.14

Zur Verbesserung des Ausdehnungsgrads darf ein neuer Block $b' = b \cup \{j\}$ immer dann gebildet werden, wenn er den Bedingungen $b \in \text{BM}(\underline{u})$, $j \in D(\underline{u})$, $g_j \leq r(b)$ und $\underline{u} \in \text{MB}$ genügt. Damit wird das Maximum $\hat{Z}$ nicht verändert. Der auf diese Weise gebildete neue Block ist, im Gegensatz zu b , stets mehrfach.

Satz 2.6.15

Sei $\underline{u} \in \text{MB}$. Sei $D(\underline{u}) \neq \emptyset$.

$$b \in 2^{D(\underline{u})} \wedge r(b) \geq 0 \Rightarrow w(b) = 0.$$

Beweis:

Angenommen $\exists b' \in 2^{D(\underline{u})}$, sodaß $w(b') > 0$ und $r(b') \geq 0$.

Dies bedeutet, daß eine optimale Zerlegung $B'(\underline{u}) = \text{BM}(\underline{u}) \cup \{b'\} \cup \{ \overset{D'(\underline{u})}{1} \}$,

$D'(\underline{u}) = D(\underline{u}) \setminus b'$, existiert, sodaß

$$\sum_{b \in B'(\underline{u})} w(b) > \hat{Z}.$$

Das steht im Widerspruch zu der Voraussetzung, daß $\hat{Z}$ das Maximum ist. $\square$

Bemerkung 2.6.16

Satz 2.6.15 bestätigt die Behauptung, daß ein Block stets den Wert 0 besitzt, wenn er eine Teilmenge der Ergänzung von $\underline{u}$ darstellt.

Bezeichnung 2.6.17

Sei $Y \subseteq V$. Mit $G(Y)$ wird das Gesamtgewicht der Knoten in Y , $\sum_{i \in Y} g_i$, bezeichnet.

Definition 2.6.18

Sei $\underline{u} \in \text{MB}$. Sei $\text{BM}(\underline{u})$ die Blockmenge von $\underline{u}$. Sei $B(\underline{u})$ die durch $\text{BM}(\underline{u})$ induzierte optimale Zerlegung von NG . $B'(\underline{u})$ heißt genau dann die von $B(\underline{u})$ abgeleitete optimale Zerlegung, wenn gilt:

$$B'(\underline{u}) \cong C1(\underline{u}) \cup C2(\underline{u}) \cup C3(\underline{u}) \cup C4(\underline{u}).$$

$$C1(\underline{u}) \cong BA(\underline{u}).$$

$$C2(\underline{u}) \cong \binom{DV(\underline{u})}{1}.$$

$$C3(\underline{u}) \subseteq \{b' \mid b' = b \cup \gamma \wedge b \in BB(\underline{u}) \wedge \gamma \in 2^{DR(\underline{u})} \wedge r(b) \geq G(\gamma)\}.$$

$$C4(\underline{u}) \subseteq \{x \mid x \in 2^{DR(\underline{u})} \setminus \{\emptyset\} \wedge G(x) \leq a\}.$$

Dabei müssen alle Elemente von $C3(\underline{u})$ und $C4(\underline{u})$ die folgenden Bedingungen erfüllen: Sei $C'(\underline{u}) = C3(\underline{u}) \cup C4(\underline{u})$.

$$\bigcup_{b \in C'(\underline{u})} b = \bigcup_{b \in BB(\underline{u})} b \cup DR(\underline{u}). \quad (1)$$

$$\forall b, b' \in C'(\underline{u}): (b \neq b' \Rightarrow b \cap b' = \emptyset). \quad (2)$$

Lemma 2.6.19

Seien $\underline{u}$, $BM(\underline{u})$, $B(\underline{u})$ und $B'(\underline{u})$ wie in Def. 2.6.18 angegeben.

- i) $|BB(\underline{u})| = |C3(\underline{u})|$.
- ii) $\left| \binom{DR(\underline{u})}{1} \right| \geq |C4(\underline{u})|$.

Beweis:

i) Angenommen $|BB(\underline{u})| \neq |C3(\underline{u})|$, dann gibt es zwei Möglichkeiten:

a) $|BB(\underline{u})| > |C3(\underline{u})|$.

Nach Def. 2.6.18 bedeutet dies, daß es mindestens ein $b \in BB(\underline{u})$ gibt, sodaß für $\forall b' \in C3(\underline{u})$ folgendes gilt:

$$b \not\subseteq b'.$$

Dies steht im Widerspruch zu der Voraussetzung, daß $b' = b \cup \{\gamma\}$.

b) $|BB(\underline{u})| < |C3(\underline{u})|$.

Nach Def. 2.6.18 bedeutet dies, daß es $b, b' \in C3(\underline{u})$ gibt, für die gilt:

$$b \neq b' \text{ und } b \cap b' \neq \emptyset.$$

Das steht im Widerspruch zu der Bedingung (2) der Definition 2.6.18.

ii) Dies folgt aus Def. 2.6.18, da alle Elemente der Menge $C4(\underline{u})$ nicht leere, miteinander disjunkte Teilmengen von $DR(\underline{u})$ sein müssen. $\square$

Satz 2.6.20

Seien $\underline{u}$, $BM(\underline{u})$, $B(\underline{u})$ und $B'(\underline{u})$ wie in Def. 2.6.18 angegeben.

- i) $|B'(\underline{u})| \leq |B(\underline{u})|$.
- ii) $\sum_{b' \in B'(\underline{u})} w(b') = \sum_{b \in B(\underline{u})} w(b) = \hat{Z}$.

Beweis:

- i) Es folgt aus Def. 2.6.8, Bezeichnung 2.6.9 und Def. 2.6.18, daß

$$|B(\underline{u})| = |BA(\underline{u})| + |BB(\underline{u})| + \left| \binom{D(\underline{u})}{1} \right| ,$$

$$|B'(\underline{u})| = |C1(\underline{u})| + |C2(\underline{u})| + |C3(\underline{u})| + |C4(\underline{u})| .$$

Dabei gilt:

$$|BA(\underline{u})| = |C1(\underline{u})| . \quad (\text{Def. 2.6.18})$$

$$|BB(\underline{u})| = |C3(\underline{u})| . \quad (\text{Lemma 2.6.19i})$$

$$\binom{D(\underline{u})}{1} = \binom{DV(\underline{u})}{1} \cup \binom{DR(\underline{u})}{1} = C2(\underline{u}) \cup \binom{DR(\underline{u})}{1} . \quad (\text{Def. 2.6.18})$$

$$\left| \binom{D(\underline{u})}{1} \right| = |C2(\underline{u})| + \left| \binom{DR(\underline{u})}{1} \right| \geq |C2(\underline{u})| + |C4(\underline{u})| . \quad (\text{Lemma 2.6.19ii})$$

Daraus folgt: $|B'(\underline{u})| \leq |B(\underline{u})|$.

- ii) Diese Aussage ergibt sich aus Def. 2.6.18, Satz 2.6.13 und Satz 2.6.15. $\square$

Korollar 2.6.21

Seien $B(\underline{u})$, $B'(\underline{u})$ wie in Def. 2.6.18 angegeben. Sei α bzw. α' der Ausdehnungsgrad von $B(\underline{u})$ bzw. von $B'(\underline{u})$.

$$\alpha \geq \alpha' .$$

Beweis:

Dies ist eine unmittelbare Folgerung des Satzes 2.6.20i). $\square$

Bemerkung 2.6.22

Ist $B(\underline{u})$ eine durch $BM(\underline{u})$ induzierte optimale Zerlegung, so besteht die Aufgabe des Rechenprozesses der Phase 3 darin, eine von $B(\underline{u})$ abgeleitete optimale Zerlegung, z.B. $B'(\underline{u})$, so zu bestimmen, daß der resultierende Ausdehnungsgrad möglichst klein wird. Da $C1(\underline{u})$ und $C2(\underline{u})$ bereits durch $BM(\underline{u})$ festgelegt sind, hängt $B'(\underline{u})$ ausschließlich von der Mächtigkeit der Teilmengen $C3(\underline{u})$ und $C4(\underline{u})$ ab. Diese beiden Teilmengen müssen vom Rechenprozeß der Phase 3 aufgefunden werden.

Definition 2.6.23

Seien $\underline{u}$, $BM(\underline{u})$, $B(\underline{u})$ und $B'(\underline{u})$ wie in Def. 2.6.18 angegeben.

Sei $BB0(\underline{u}) = BB(\underline{u}) \cup \{\emptyset\}$. Ein Ergänzungsprozeß auf $B(\underline{u})$ ist ein Tripel

$$E郑 = (BB0(\underline{u}), DR(\underline{u}), \Psi),$$

wobei Ψ eine Abbildung ist:

$$\Psi: BB0(\underline{u}) \times 2^{DR(\underline{u})} \rightarrow 2^V,$$

angegeben durch:

$$\Psi(b, X) = \begin{cases} b \cup X, & \text{wenn } r(b) \geq G(X). \\ \text{undefiniert,} & \text{wenn } r(b) < G(X). \end{cases}$$

$\Psi(b, X)$ ist g.d. der (durch X) ergänzte Block, wenn gilt:

- i) $\forall b' \forall X' (b \neq b' \wedge X \neq X' \Rightarrow \Psi(b, X) \cap \Psi(b', X') = \emptyset).$
 ii) $\bigcup_{\substack{b \in BB0(\underline{u}) \\ X \in 2^{DR(\underline{u})}}} \Psi(b, X) = \bigcup_{b \in BB(\underline{u})} b \cup DR(\underline{u}).$

Satz 2.6.24

Seien $\underline{u}$, $BM(\underline{u})$, $B(\underline{u})$ und $B'(\underline{u})$ wie in Def. 2.6.18 angegeben.

Sei $E郑 = (BB0(\underline{u}), DR(\underline{u}), \Psi)$ ein Ergänzungsprozeß auf $B(\underline{u})$.

Sei $C'(\underline{u}) = C3(\underline{u}) \cup C4(\underline{u})$. Sei $b' \in B'(\underline{u})$.

$$b' \in C'(\underline{u}) \Rightarrow b' \text{ ist ein ergänzter Block.}$$

Beweis:

- $b' \in C'(u)$
 $\Rightarrow b' \in C3(u) \cup C4(u)$, wobei b' den Bedingungen der Def. 2.6.18 genügt
 $\Rightarrow \exists b \in BB0(u) \wedge \exists x \in 2^{DR(u)}$, sodaß $b' = \Psi(b, X) = b \cup X$
 und $\Psi(b, X)$ genügt den Bedingungen i) und ii) der Def. 2.6.23
 $\Rightarrow b'$ ist ein ergänzter Block. $\square$

Wir wollen nun zwei Algorithmen angeben, die es ermöglichen, eine Menge der ergänzten Blöcke von $BB(u)$ mit wenig Aufwand aufzufinden. Die Algorithmen beruhen auf zwei bekannten heuristischen Strategien: FFD ("first-fit decreasing") und BFD ("best-fit decreasing"). Diese beiden Strategien können zwar kein Ergebnis mit minimalem Ausdehnungsgrad garantieren, sie ermöglichen es jedoch, sehr schnell eine $B'(u)$ aufzufinden, deren Mächtigkeit in den meisten Fällen dem Minimum sehr nahe kommt. Die noch anzugebenden Algorithmen unterscheiden sich von den üblichen FFD- (bzw. BFD-) Algorithmen im wesentlichen dadurch, daß $r(b)$ keine Konstante ist, sondern Werte besitzt, die der Relation $0 \leq r(b) \leq a$ genügen.

Bezeichnung 2.6.25

Seien $i, j, p, q, l, m \in \mathbb{N}$. Sei $u \in ZB$. Sei $BB(u) = \{b_i \mid i \in \mathbb{N}_m\}$, wobei gilt:
 $\forall b_i, b_j \in BB(u): (i \leq j \Rightarrow r(b_i) \leq r(b_j))$. Sei $DR(u) = \{k_p \mid p \in \mathbb{N}_l\}$, wobei gilt:
 $\forall k_p, k_q \in DR(u): (p \leq q \Rightarrow g(k_p) \geq g(k_q))$. $g(k_p)$ (bzw. $g(k_q)$) bezeichnet das Gewicht des Knotens k_p (bzw. k_q). Mit C_R wird die Folge $(r_1, \dots, r_m)$ bezeichnet, wobei $r_i = r(b_i)$ für $\forall i \in \mathbb{N}_m$. Mit C_G wird die Folge $(g_1, \dots, g_l)$ bezeichnet, wobei $g_p = g(k_p)$ für $\forall p \in \mathbb{N}_l$. Mit B wird die Folge $(b_1, \dots, b_m)$ bezeichnet. Als elementare Operationen werden alle arithmetischen Operationen bzw. alle logischen Verknüpfungen und Operationen für Mengen bezeichnet.

Algorithmus 2.6.26

Name: "FFD"

Voraussetzung: B , $DR(u)$, C_R und C_G sind gegeben.

$m = |BB(u)|$, $l = |DR(u)|$, a ist die Gewichtsgrenze.

i, j, x, Y, A, H sind Variable.

Startbedingung: $H := \emptyset$, $A := \emptyset$, $Y := C_G$, $x := 0$, $i := 1$.

Iteration:

1. $A := b_i$, $x := r_i$.
2. Suche eine Komponente y_j von Y , deren Index $j \in \mathbb{N}_1$ am kleinsten ist, sodaß $0 < y_j \leq x$.

Wird y_j gefunden, dann

$$A := A \cup \{k_j\}$$

$$x := x - y_j$$

und Sprung nach 3.

Falls es kein $j \in \mathbb{N}_1$ gibt, sodaß $0 < y_j \leq x$, Sprung nach 4.

3. $y_j := 0$

Falls $x \neq 0$, wiederhole den Schritt 2.

4. $H := H \cup \{A\}$.

Falls $i < m \wedge Y \neq \underline{0}$, dann $i := i + 1$ und Sprung nach 1.

Falls $i < m \wedge Y = \underline{0}$, dann Sprung nach 5.

Falls $i \geq m \wedge Y \neq \underline{0}$, dann $i := i + 1$ und Sprung nach 6.

Falls $i \geq m \wedge Y = \underline{0}$, dann Sprung nach 7.

5. $H := H \cup \bigcup_{j=i+1}^m \{b_j\}$, Sprung nach 7.

6. $A := \emptyset$, $x := a$, Sprung nach 2.

7. Ende.

Ergebnis: $H = C'(\underline{u})$, wobei $C'(\underline{u}) = C3(\underline{u}) \cup C4(\underline{u})$.

Satz 2.6.27

FFD bricht nach höchstens $(m + 1)$ Iterationen ab. Jede Iteration benötigt $O(1)$ elementare Operationen. Nach Abbruch des Algorithmus gilt $H = C3(\underline{u}) \cup C4(\underline{u})$.

Beweis:

Bei jeder Iteration wird der Index i um eins erhöht. Dies geschieht bei Schritt 4 in den folgenden zwei Fällen:

i) $i < m \wedge Y \neq \underline{0}$.

ii) $i \geq m \wedge Y \neq \underline{0}$.

In den anderen zwei Fällen, d.h. $i < m \wedge Y = \underline{0}$ oder $i \geq m \wedge Y = \underline{0}$, bricht der Algorithmus bei Schritt 7 ab.

Der Fall i) benötigt höchstens m Iterationen. Danach tritt der Fall ii) ein, der höchstens 1 Iterationen benötigt. Wenn die Bedingung $g_j > r_i \wedge \frac{1}{2}a < g_j \leq a$ für $i \in \mathbb{N}_m$ und $j \in \mathbb{N}_1$ erfüllt wird, dann bricht der Algorithmus nach genau $m + 1$ Iterationen ab.

Die Anzahl der elementaren Operationen, die bei einer Iteration ausgeführt werden, wird im wesentlichen von der Anzahl der Such- und Vergleichsoperationen des Schritts 2 bestimmt. Sie nimmt proportional mit der Anzahl der Komponenten der Folge Y zu. Eine Iteration benötigt also $O(1)$ elementare Operationen.

Solange die Bedingung $i < m$ erfüllt wird, werden bei jeder Iteration die Schritte 1, 2, (3) und 4 ausgeführt. Bei Schritt 2, der sich von Schritt 3 aus mehrmals wiederholen kann, wird jeweils ein Element der Menge $C3(\underline{u})$ aufgefunden. Die Iteration wird solange fortgesetzt, bis entweder $Y = \underline{0}$ oder $i \geq m$ wird.

Ist $Y = \underline{0}$, dann bricht FFD nach der Ausführung des Schritts 5 ab. In diesem Falle gilt: $H = C3(\underline{u})$ und $C4(\underline{u}) = \emptyset$.

Gilt $i \geq m \wedge Y \neq \underline{0}$, dann wird bei jeder Iteration, bei der die Schritte 6, 2, (3) und 4 ausgeführt werden, jeweils ein Element der Menge $C4(\underline{u})$ festgelegt. Der Algorithmus bricht ab, wenn $Y = \underline{0}$ wird, d.h. wenn alle Elemente der Menge $C4(\underline{u})$ aufgefunden worden sind. In diesem Falle gilt $H = C3(\underline{u}) \cup C4(\underline{u})$. $\square$

Algorithmus 2.6.28

Name: "BFD"

Voraussetzung: B , $DR(\underline{u})$, C_R und C_G sind gegeben.

$m = |BB(\underline{u})|$, $l = |DR(\underline{u})|$, a ist die Gewichtsgrenze.

$A = (a_1, \dots, a_p)$, $Y = (y_1, \dots, y_p)$ für $p \in \mathbb{N}$.

i, j, n, x, p und H sind Variable.

Startbedingung: $H := \emptyset$, $A := B$, $Y := C_R$, $n := m$, $x := 0$, $i := 1$.

Iteration:

1. $x := g_i$.
2. Suche ein y_j aus Y , sodaß $(y_j - x) \geq 0$ und $(y_j - x)$ für $\forall j \in \mathbb{N}_n$ minimal ist.
 Wenn es kein $j \in \mathbb{N}_n$ gibt, sodaß $y_j - x \geq 0$, dann $n := n + 1$ und Sprung nach 4.
 Wird y_j gefunden, dann $a_j := a_j \cup \{k_i\}$ und $y_j := y_j - x$.
3. Wenn $i < l$, dann $i := i + 1$ und Sprung nach 1.
 Wenn $i = l$, dann

$$H := H \cup \bigcup_{j=1}^n \{a_j\}$$
 und Sprung nach 6.
4. $j := n$.
5. $a_j := \emptyset$, $y_j := a$ und Sprung nach 2.
6. Ende.

Ergebnis: $H = C3(\underline{u}) \cup C4(\underline{u})$.

Satz 2.6.29

BFD bricht nach l Iterationen ab. Jede Iteration benötigt höchstens $O(m+1)$ elementare Operationen. Nach Abbruch des Algorithmus gilt $H = C3(\underline{u}) \cup C4(\underline{u})$.

Beweis:

Bei Schritt 3 wird der Index i bei jeder Iteration um eins erhöht, bis $i = l$ wird, wobei der Algorithmus abbricht.

Bei jeder Iteration werden die Schritte 1, 2, 3 oder die Schritte 1, 2, 4, 5 ausgeführt. Die Anzahl der elementaren Operationen einer Iteration wird im wesentlichen von der Anzahl derer, die zur Auffindung eines minimalen $(y_j - x) \geq 0$ benötigt werden, bestimmt. Die Anzahl der elementaren Operationen zur Bestimmung des Minimums einer Liste der Länge n ist bekanntlich $O(n)$ (z.B. /93/), wobei n für BFD höchstens $(m + 1)$ beträgt.

Beim Start des Algorithmus gilt $a_j = b_j$ für $\forall j \in \mathbb{N}_m$. Bei Schritt 2 wird a_j mit einer Teilmenge von $DR(\underline{u})$ vereinigt, wenn die Bedingung $(r(a_j) \geq g_i \wedge r(a_j) - g_i \text{ ist minimal})$ erfüllt ist. Die derart ergänzten Blöcke bilden die Menge $C3(\underline{u})$.

Die Elemente von $C4(\underline{u})$ werden bei der Iteration 1-2-(3)-4-5 gebildet. Jedes Element a_j , $j > m$, erhält bei Schritt 5 den Anfangswert $\emptyset$. Es wird durch eine Teilmenge von $DR(\underline{u})$ ersetzt, falls bei Schritt 2 ein y_j gefunden wird, das der Relation $y_j - x \geq 0$ genügt. Beim Abbruch des Algorithmus gilt $H = C3(\underline{u}) \cup C4(\underline{u})$. $\square$

Beispiel 2.6.30

Wir können nun die endgültige Lösung des Beispiels 2.3.45 erhalten:

$$\begin{aligned}\underline{u} &= (1, 0, 0, 1, 0, 1, 0, 0, 0, 0) && (\text{Beispiel 2.5.17}) \\ BM(\underline{u}) &= \{\{2, 3\}, \{5, 6\}, \{8, 9, 10\}\} && (\text{Def. 2.6.1}) \\ D(\underline{u}) &= \{1, 4, 7, 11, 12\} && (\text{Def. 2.6.7}) \\ DV(\underline{u}) &= \emptyset && (\text{Bezeichnung 2.6.9}) \\ DR(\underline{u}) &= \{1, 4, 7, 11, 12\} && (\text{Bezeichnung 2.6.9}) \\ \gamma &= 4. && (\text{Def. 2.6.7})\end{aligned}$$

Die durch $BM(\underline{u})$ induzierte optimale Zerlegung ist:

$$B(\underline{u}) = BA(\underline{u}) \cup BB(\underline{u}) \cup \binom{D(\underline{u})}{1}, \quad (\text{Def. 2.6.8 und Bezeichnung 2.6.9})$$

wobei gilt:

$$\begin{aligned}BA(\underline{u}) &= \{\{2, 3\}, \{5, 6\}, \{8, 9, 10\}\}. \\ BB(\underline{u}) &= \emptyset. \\ \binom{D(\underline{u})}{1} &= \{\{1\}, \{4\}, \{7\}, \{11\}, \{12\}\}.\end{aligned}$$

Die von $B(\underline{u})$ abgeleitete optimale Zerlegung ist:

$$B'(\underline{u}) = C1(\underline{u}) \cup C2(\underline{u}) \cup C3(\underline{u}) \cup C4(\underline{u}), \quad (\text{Def. 2.6.18})$$

wobei gilt:

$$\begin{aligned}C1(\underline{u}) &= \{\{2, 3\}, \{5, 6\}, \{8, 9, 10\}\}. \\ C2(\underline{u}) &= \emptyset. \\ C3(\underline{u}) &= \emptyset. \\ C4(\underline{u}) &= \{\{1\}, \{4\}, \{7\}, \{11\}, \{12\}\}.\end{aligned}$$

Die endgültige Lösung des Beispiels 2.3.45 ist $(B'(\underline{u}), 165)$. Für dieses spezielle Beispiel gilt $B(\underline{u}) = B'(\underline{u})$, unabhängig davon, ob FFD oder BFD benutzt wird. Der Ausdehnungsgrad lautet: $\alpha = 8/7$.

2.7 Eine zusammenfassende Darstellung des Angsches Rechenverfahrens zur optimalen Zerlegung von Netzen

In Bild 2.7.1 werden das in Abschnitt 2.2 ausgearbeitete Konzept zur optimalen Zerlegung von Netzen und die zur Lösung der einzelnen Teilprobleme entwickelten Rechenprozesse schematisch dargestellt. Die Erläuterung der einzelnen Symbole befindet sich in der Tabelle 2.7.2. Das Konzept und die Rechenprozesse zusammen werden künftig Angsches Rechenverfahren zur optimalen Zerlegung von Netzen genannt.

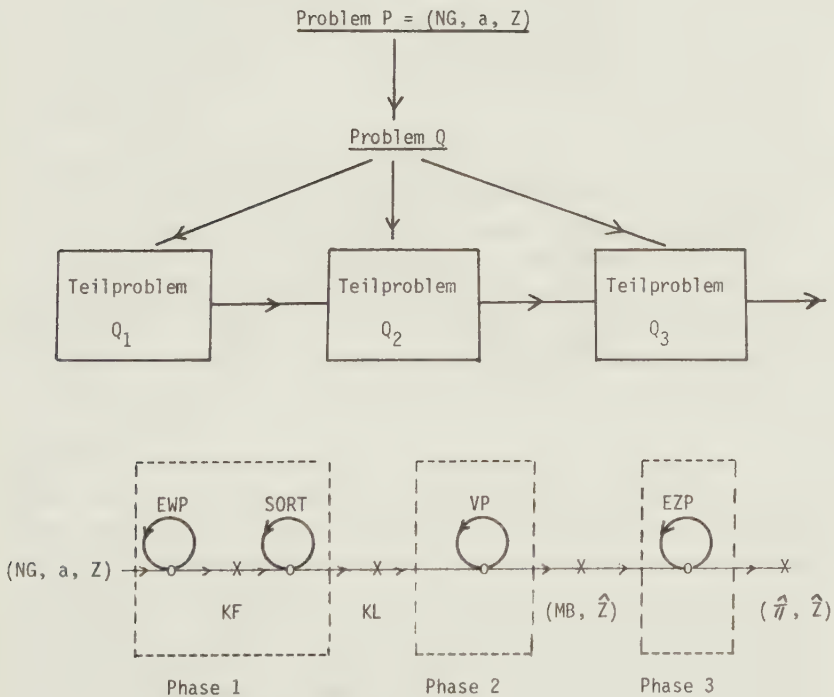


Bild 2.7.1

<u>Symbol</u>	<u>Bezeichnung</u>	<u>Bedeutung</u>
	Rechenprozeß	
$\ast$	Ergebnis eines Rechenprozesses	
NG	ungerichtetes Netz	Def. 2.1.5
a	Gewichtsgrenze	Def. 2.1.6
Z	Zielfunktion	Def. 2.1.6
$\hat{Z}$	Optimum (maximum)	Def. 2.1.6
Q_1, Q_2, Q_3	Teilprobleme des Problems Q	Def. 2.2.6 und Seite 39
$\hat{\pi}$	Menge der optimalen Zerlegungen	Def. 2.1.6
EW P	Erweiterungsprozeß	Def. 2.5.30
KF	Kern der Indikatormenge	Def. 2.5.13
SORT	Prozedur "SORT"	Algorithmus 2.3.41
KL	K-Liste	Def. 2.5.13
VP	Binärer Verzweigungsprozeß	Def. 2.4.24
MB	Menge aller maximalen Blätter einer binären Verzweigungs- struktur	Def. 2.4.14
EZP	Ergänzungsprozeß	Def. 2.6.23

Tabelle 2.7.2

2.8 Problembeispiele und Bemerkungen zum Rechenaufwand

Drei weitere Probleme werden als zusätzliches Beispiel mittels des Angschen Rechenverfahrens gelöst. Diese Probleme und ihre Lösungen sind in Anhang D angegeben, in dem auch die K-Listen der Netze und die schematische Darstellung der binären Verzweigungsprozesse der entsprechenden Probleme zu finden sind. Über die Herkunft dieser Probleme sei folgendes bemerkt:

Problem 2.8.1

In diesem Problem wird ein Netz angegeben, das aus 35 Knoten besteht. Die Struktur des Netzes und dessen Kantenbewertung stammt von Herrn Schröck /128/ der Universität Bonn. Die Gewichtsgrenze und die Kantengewichte sind dem Beispiel von Baer und Caughey /113/ entnommen. Da das Netz von Baer und Caughey nur 24 Knoten enthält, wurden die übrigen Knotengewichte (für die Knoten 25 - 35) gewürfelt. Die manuelle Rechenzeit beträgt ca. 6 Stunden.

Problem 2.8.2

Dieses Problem ist ein Beispiel, das in der Arbeit von B. W. Kernighan /118/ enthalten ist. In dieser Arbeit wird ein Rechenverfahren zur sequentiellen Partition von Netzen angegeben, bei dem einschränkend vorausgesetzt wird, daß die Indizes der in einem Block zusammengefaßten Knoten fortlaufende Nummern sein müssen (deshalb wird das Verfahren als "sequentielle Partition" bezeichnet). Es hat sich herausgestellt, daß dieses Beispiel vom Autor /118/ so gewählt wurde, daß sich aus der sequentiellen Partition tatsächlich eine optimale Zerlegung ergibt. Die manuelle Rechenzeit beträgt ca. 3 Stunden.

Problem 2.8.3

Das Netz ist ein Programmgraph, der sich aus der Messung des dynamischen Programmverhaltens im Adreßraum ergibt. Einzelheiten über das Meßverfahren und über das Programm, dessen Verhalten durch diesen Programmgraphen dargestellt wird, finden sich in /175/. Eine spezielle Eigenschaft dieses Beispiels besteht darin, daß die Knoten des in /175/ definierten Programmgraphen Mengen von Seiten sind, die nicht paarweise miteinander disjunkt zu sein brauchen. Es muß also dafür gesorgt werden, daß man am Ende der Phase 1 eine K-Liste erhält, deren Elemente (die Blockindikatoren) sich auf die Seiten (anstatt auf die Knoten des Programmgraphen) beziehen. Als Gewichtsgrenze wird die Größe eines Programmsegments angegeben. Das Netz enthält 29 Knoten. Die manuelle Rechenzeit beträgt ca. 8 Stunden.

Die Erfahrungen, die man bei der Lösung dieser Problembeispiele gesammelt hat, sind noch zu gering, um eine allgemeingültige Aussage über den Aufwand des Angschen Rechenverfahrens zu ermitteln. Die anfängliche Befürchtung, daß die Anzahl der Elemente der K-Liste exponentiell mit $|V|$ zunehmen könnte, hat sich bei der Lösung der vorliegenden Problembeispiele nicht bestätigt. Die Anzahl m der Elemente der K-Liste für die meisten in der Praxis auftretenden Probleme scheint eher der Relation $0 \leq m \leq c|V|$ zu genügen, wobei $c \geq 1$ eine Konstante ist, die von der Struktur des Netzes und von der relativen Blockkapazität bestimmt wird.

Die bisher angesammelten Rechenerfahrungen beruhen ausschließlich auf dem manuellen Rechnen. Der Versuch, Problembeispiele mittels des Angschen Rechenverfahrens manuell zu lösen, hat insbesondere während der Entwicklungsphase den Vorteil, daß man die einzelnen Rechenschritte bzw. Zwischenergebnisse sieht und darüber nachdenkt. Dadurch gewinnt man nicht nur Einsicht in den Ablauf des ganzen Rechenprozesses, sondern auch viele neuen Ideen dazu, die zur Verbesserung des Rechenverfahrens und zur mathematischen Begründung der Algorithmen beitragen könnten. Zum Vergleich betrachten wir die Arbeit von E. Balas /43/ (siehe auch /44/), in der der bewährte "additive Algorithmus" zur Lösung linearer Programmierungsaufgaben mit 0 - 1 Variablen beschrieben und begründet wird. Darin sind 11 Problembeispiele mit 5 bis 15 Variablen angegeben, die im Durchschnitt jeweils 1 - 3 Stunden manuelle Rechenzeit benötigen. Die Brauchbarkeit des additiven Algorithmus zur Lösung praktischer Probleme wurde durch zahlreiche Experimente (z.B. /50, 51/) bestätigt, bei denen der Algorithmus programmiert und getestet wurde. Geht man davon aus, daß der Rechenaufwand ausschließlich von der Anzahl der Variablen bestimmt wird, die in einem Problem enthalten sind, so wird man feststellen können, daß die Rechenzeit des Angschen Verfahrens zur Lösung der vorliegenden drei Probleme - die im Grunde quadratische Optimierungsaufgaben sind (Def. 2.1.22) - größenordnungsmäßig durchaus vergleichbar mit der Rechenzeit des additiven Algorithmus ist. Berücksichtigt man außerdem die Tatsache, daß ein quadratisches Optimierungsproblem im allgemeinen sehr viel schwerer zu lösen ist als ein lineares Programmierungsproblem mit der gleichen Anzahl von Variablen, so gewinnt das Angsche Rechenverfahren deutliche Vorteile in Bezug auf die Rechenzeit.

Da die wesentlichsten Rechenschritte des Angschen Rechenverfahrens aus Suchoperationen und logischen Verknüpfungen von boolischen Vektoren bestehen, eignet es sich besonders gut für die praktische Implementierung in einer

Rechenanlage. Der Speicherplatzbedarf hängt von dem Prozeß ab, der gerade ausgeführt wird. Die Prozesse der Phase 1 haben den größten Speicherplatzbedarf, da alle Knoten der markierten Erweiterungsbäume im Arbeitsspeicher bzw. im Hintergrundspeicher beibehalten werden müssen. Aus diesen Knoten wird dann die K-Liste gebildet. Der binäre Verzweigungsprozeß benötigt verhältnismäßig wenig Speicherplatz, da zu jedem Zeitpunkt der Laufphase nur ein Zweig der binären Verzweigungsstruktur für den Gang des binären Verzweigungsprozesses benötigt wird. Der Speicheraufwand des Ergänzungsprozesses ist im Vergleich zu dem der anderen Prozesse sehr gering.

Ein zusätzlicher Vorteil des Angsches Rechenverfahrens liegt in seiner Flexibilität bei der Lösung sehr großer Probleme, wobei die gegebene Rechenzeit und Speicherkapazität nicht überschritten werden darf. Für die Zerlegung von Netzen mit Hunderten von Knoten und mehr liefert das Verfahren unter dieser Bedingung eine approximierende Lösung mit berechneten oder mit vorgegebenen Fehlerschranken. Die vom Angschen Rechenverfahren abgeleiteten Annäherungsverfahren werden in Kapitel 3 ausführlich beschrieben und begründet.

3. Die optimale Zerlegung von Netzen - deterministische Approximation.

3.1 Einführung

In diesem Kapitel werden wir deterministische Annäherungsverfahren untersuchen, die auf dem Angschen Rechenverfahren beruhen. Diese Annäherungsverfahren benötigen im allgemeinen viel geringeren Rechenaufwand als das exakte Rechenverfahren. An erster Stelle interessieren wir uns für einen binären Verzweigungsprozeß, der in der Lage sein soll, eine approximierende Lösung mit vorgegebenen oder mit berechneten Fehlerschranken zu liefern. Die Angabe von Fehlerschranken hat den Vorteil, daß die Güte der approximierenden Lösung dadurch garantiert wird. Der approximierende binäre Verzweigungsprozeß (Def. 2.4.24) spielt für diese Art von Approximation die wichtigste Rolle. Anschließend werden Annäherungsverfahren untersucht, die sich zur Lösung von solchen Problemen besonders gut eignen, bei denen zusätzliche Bedingungen erfüllt werden müssen. Hierzu gehören Probleme, die sowohl eine obere als auch eine untere Gewichtsgrenze besitzen, oder Probleme, bei denen die Länge der K-Liste nicht größer als ein gegebener Grenzwert sein darf (z.B. wegen der beschränkten Speicherkapazität). Für solche Probleme werden Methoden auf der Grundlage des Angschen Rechenverfahrens entwickelt, die es ermöglichen, entweder eine exakte Lösung oder eine gute Approximation zu liefern.

3.2 Ein Annäherungsverfahren, bei dem der relative Fehler vorgegeben ist.

Es wird in diesem Abschnitt angenommen, daß die Phase 1 des Angschen Rechenverfahrens bereits abgeschlossen ist und die entsprechende K-Liste vorliegt. Von dieser K-Liste ausgehend werden wir die Eigenschaften des approximierenden binären Verzweigungsprozesses untersuchen. Seien $VST = (P, A, KF, \varphi, \omega, \psi, \bar{\varphi})$, $SU = (MR, AS)$ und $MR = (KM, \xi, \delta)$. Sei $VP = (VST, SU, \varepsilon)$ ein binärer Verzweigungsprozeß.

Definition 3.2.1

Eine Suchstrategie $SU = (MR, AS)$ heißt genau dann ξ -approximierend, wenn gilt:

i) $MR = (KM, \xi, \delta)$ ist eine Markierungsregel (Def. 2.4.21), für die gilt:

$$KM = \{L, R, B, A\}.$$

$$\xi : P \times KM \longrightarrow KM.$$

$$\delta : P \times KM \rightarrow P.$$

ii) AS ist die Anfangsschranke (Def. 2.4.22).

iii) Es gibt ein $\varepsilon \in [0, 1)$, sodaß ξ und δ folgendermaßen angegeben werden (vgl. Def. 2.4.41):

Fall 1: $\underline{u} = \emptyset \wedge \exists v((\underline{u}, \underline{v}) \in A)$.

$(\underline{u}, \underline{\lambda}) \in P \times KM \implies \delta(\underline{u}, \underline{\lambda}) = \underline{v} \wedge \xi(\underline{u}, \underline{\lambda}) = R \wedge \underline{v} = \text{Ja-Sohn von } \underline{u}$.

$(\underline{u}, R) \in P \times KM \implies \delta(\underline{u}, R) = \underline{v} \wedge \xi(\underline{u}, R) = L \wedge \underline{v} = \text{Nein-Sohn von } \underline{u}$.

$(\underline{u}, L) \in P \times KM \implies \delta(\underline{u}, L) = \underline{u} \wedge \xi(\underline{u}, L) = B$.

$(\underline{u}, B) \in P \times KM \implies \delta(\underline{u}, B) = \underline{u} \wedge \xi(\underline{u}, B) = B$.

Fall 2: $\underline{u} \neq \emptyset \wedge \exists F_j \exists v \exists t (\mathcal{G}(\underline{u}) \in F_j \wedge (\underline{u}, \underline{v}) \in A \wedge (\underline{t}, \underline{u}) \in A)$.

Fall 2.1: $\underline{u} \in ZL \wedge \omega(\underline{u}) + \tilde{\varphi}(\underline{u}) > US \cdot (1 + \varepsilon) \wedge \mathcal{G}(\underline{v}) \in F_j \wedge ((\underline{u} = \text{Ja-Knoten}) \vee (\underline{u} = \text{Nein-Knoten} \wedge \exists \underline{s} (\underline{s} \in ZL \wedge \mathcal{G}(\underline{s}) \in F_j \wedge \underline{s} = \text{Vorgänger von } \underline{u} \wedge \underline{s} = \text{Ja-Knoten})))$.

$(\underline{u}, \underline{\lambda}) \in P \times KM \implies \delta(\underline{u}, \underline{\lambda}) = \underline{v} \wedge \xi(\underline{u}, \underline{\lambda}) = L \wedge \underline{v} = \text{Nein-Sohn von } \underline{u}$.

$(\underline{u}, L) \in P \times KM \implies \delta(\underline{u}, L) = \underline{t} \wedge \xi(\underline{u}, L) = B$.

Für $(\underline{u}, R)$ und $(\underline{u}, B)$ sind ξ und δ undefiniert.

Fall 2.2: $\underline{u} \in ZL \wedge \omega(\underline{u}) + \tilde{\varphi}(\underline{u}) > US \cdot (1 + \varepsilon) \wedge ((\mathcal{G}(\underline{v}) \notin F_j) \vee (\mathcal{G}(\underline{v}) \in F_j \wedge \underline{u} = \text{Nein-Knoten} \wedge \forall \underline{s} (\mathcal{G}(\underline{s}) \in F_j \wedge \underline{s} = \text{Vorgänger von } \underline{u} \wedge \underline{s} = \text{Nein-Knoten})))$.

$(\underline{u}, \underline{\lambda}) \in P \times KM \implies \delta(\underline{u}, \underline{\lambda}) = \underline{v} \wedge \xi(\underline{u}, \underline{\lambda}) = R \wedge \underline{v} = \text{Ja-Sohn von } \underline{u}$.

$(\underline{u}, R) \in P \times KM \implies \delta(\underline{u}, R) = \underline{v} \wedge \xi(\underline{u}, R) = L \wedge \underline{v} = \text{Nein-Sohn von } \underline{u}$.

$(\underline{u}, L) \in P \times KM \implies \delta(\underline{u}, L) = \underline{t} \wedge \xi(\underline{u}, L) = B$.

Für $(\underline{u}, B)$ sind ξ und δ undefiniert.

Fall 2.3: $(\underline{u} \in ZL \wedge \omega(\underline{u}) + \tilde{\varphi}(\underline{u}) \leq US \cdot (1 + \varepsilon)) \vee (\underline{u} \in UZL)$.

$(\underline{u}, \underline{\lambda}) \in P \times KM \implies \delta(\underline{u}, \underline{\lambda}) = \underline{t} \wedge \xi(\underline{u}, \underline{\lambda}) = B$.

Für $(\underline{u}, R)$, $(\underline{u}, L)$ und $(\underline{u}, B)$ sind ξ und δ undefiniert.

Fall 3: $\underline{u}$ ist ein Blatt $\wedge (\underline{t}, \underline{u}) \in A$.

$(\underline{u}, \underline{\lambda}) \in P \times KM \implies \delta(\underline{u}, \underline{\lambda}) = \underline{t} \wedge \xi(\underline{u}, \underline{\lambda}) = B$.

Für $(\underline{u}, R)$, $(\underline{u}, L)$ und $(\underline{u}, B)$ sind ξ und δ undefiniert.

$VP = (VST, SU, \varepsilon)$ heißt genau dann $\underline{\varepsilon}$ -approximierend, wenn $\mathcal{E}(US) = \underline{\varepsilon} > 0$ eine Konstante und SU eine $\underline{\varepsilon}$ -approximierende Suchstrategie ist (vgl. Def. 2.4.24).

Bemerkung 3.2.2

Eine ε -approximierende Suchstrategie ist im Grunde die Suchstrategie (MARZ, AS), wobei die Bedingung $\omega(\underline{u}) + \bar{\varphi}(\underline{u}) > US$ (bzw. $\omega(\underline{u}) + \bar{\varphi}(\underline{u}) \leq US$) durch $\omega(\underline{u}) + \bar{\varphi}(\underline{u}) > US \star (1 + \varepsilon)$ (bzw. $\omega(\underline{u}) + \bar{\varphi}(\underline{u}) \leq US \star (1 + \varepsilon)$) ersetzt wird. Falls $\varepsilon = 0$, dann ist die ε -approximierende Suchstrategie identisch mit (MARZ, AS). Die ε -approximierende Suchstrategie läßt sich also als eine Verallgemeinerung von (MARZ, AS) betrachten. Ein exakter VP ist gleichbedeutend mit einem ε -approximierenden VP, bei dem $\varepsilon = 0$ gilt.

Satz 2.4.33 in Abschnitt 2.4 läßt sich für $VP = (VST, SU, \varepsilon)$ folgendermaßen verallgemeinern:

Satz 3.2.3

Sei $[0, t]$ mit $t \in \mathbb{R}^+$ eine Laufphase des binären Verzweigungsprozesses $VP = (VST, SU, \varepsilon)$. Seien $i, j \in \mathbb{N}$ und $t_i, t_j \in [0, t]$. Seien $US(t_i), US(t_j)$ die unteren Schranken zum Zeitpunkt t_i bzw. t_j .

- i) $t_i \geq t_j \Rightarrow US(t_i) \geq US(t_j)$.
 ii) $\hat{Z} \star (1 - \varepsilon) \leq US(t) \leq \hat{Z}$.

Beweis:

- i) Diese Aussage ist identisch mit der Aussage i) des Satzes 2.4.33.
 ii) Offensichtlich gilt $US(t) \leq \hat{Z}$, sonst wäre $US(t) > \hat{Z}$, ein Widerspruch. Es bleibt zu beweisen, daß $\hat{Z} \star (1 - \varepsilon) \leq US(t)$.

Sei $t_k \in [0, t]$, $k \in \mathbb{N}$, der Zeitpunkt, zu dem der laufende VP die untere Schranke $US(t_k) = US(t)$ erhält. Nach Def. 2.4.24 bedeutet dies, daß im Zeitintervall $(t_k, t]$ die untere Schranke von VP unverändert bleibt. Dieser Fall kann nur eintreten, wenn der VP im Zeitintervall $(t_k, t]$ kein Blatt $\underline{v}$ markiert, bei dem gilt $\omega(\underline{v}) > US(t)$. Nach Def. 3.2.1 bedeutet dies, daß es für $\forall \underline{v} \in ZB$, $\omega(\underline{v}) > US(t)$, einen echten Vorgänger $\underline{u} \in ZL$ gibt, sodaß

$$\omega(\underline{u}) + \bar{\varphi}(\underline{u}) \leq US(t) \star (1 + \varepsilon). \quad (\text{Def. 3.2.1, Fall 2.3})$$

$$\text{Da} \quad \omega(\underline{u}) \leq \omega(\underline{v}) \leq \omega(\underline{u}) + \bar{\varphi}(\underline{u}), \quad (\text{Def. 2.4.12})$$

$$\text{so folgt} \quad \omega(\underline{v}) - US(t) \star \varepsilon \leq US(t).$$

Es gilt insbesondere für $\underline{v} \in \text{MB}$ (d.h. $\omega(\underline{v}) = \hat{Z}$):

$$\hat{Z} - \text{US}(\underline{t}) \cdot \varepsilon \leq \text{US}(\underline{t}).$$

$$\hat{Z} - \hat{Z} \cdot \varepsilon \leq \text{US}(\underline{t}) \quad (\text{US}(\underline{t}) \leq \hat{Z})$$

$$\hat{Z} \cdot (1 - \varepsilon) \leq \text{US}(\underline{t}). \quad \square$$

Bezeichnung 3.2.4

Sei $\underline{v} \in \text{ZB}$. $(\underline{v}, \omega(\underline{v}), \varepsilon)$ bezeichnet ein ε -approximierendes Ergebnis von VP genau dann, wenn $\omega(\underline{v})$ der folgenden Relation genügt:

$$\hat{Z} \cdot (1 - \varepsilon) \leq \omega(\underline{v}) \leq \hat{Z}.$$

Jedes Annäherungsverfahren, das ein ε -approximierendes Ergebnis garantiert, läßt sich als ε -approximierender Algorithmus im Sinne von Sahni /90, 108/ betrachten.

Satz 3.2.5

Seien $\text{VP}_1 = (\text{VST}, \text{SU}, \varepsilon_1)$, $\text{VP}_2 = (\text{VST}, \text{SU}, \varepsilon_2)$. Sei $\text{AW}(\text{VP}_1)$ bzw. $\text{AW}(\text{VP}_2)$ der Aufwand von VP_1 bzw. von VP_2 .

$$\varepsilon_1 < \varepsilon_2 \implies \text{AW}(\text{VP}_1) \geq \text{AW}(\text{VP}_2).$$

Beweis:

Bei der Suchstrategie SU wird die Markierungsregel durch die Vorschriften in Def. 3.2.1 angegeben. ε spielt ausschließlich in Fall 2 der Vorschriften eine Rolle. In diesem Falle wird bei jedem Knoten $\underline{u}$, der markiert werden soll, die Bedingung $\omega(\underline{u}) + \underline{e}(\underline{u}) > \text{US} \cdot (1 + \varepsilon)$ geprüft. Ist diese Bedingung erfüllt, so macht der VP einen Vorwärtsschritt. Ist sie nicht erfüllt, dann folgt ein Rückwärtsschritt des VP. Bei einem Vorwärtsschritt wird der Aufwand im Vergleich zum Rückwärtsschritt um mindestens zwei Schritte erhöht. Gilt $\varepsilon_1 < \varepsilon_2$, dann hat VP_1 bessere Chancen als VP_2 , von $\underline{u}$ aus weitere Vorwärtsschritte zu machen. Da dies für $\forall \underline{u} \in P$ gilt, die weder Wurzel noch Blätter von VST sind, so folgt die zu beweisende Aussage. $\square$

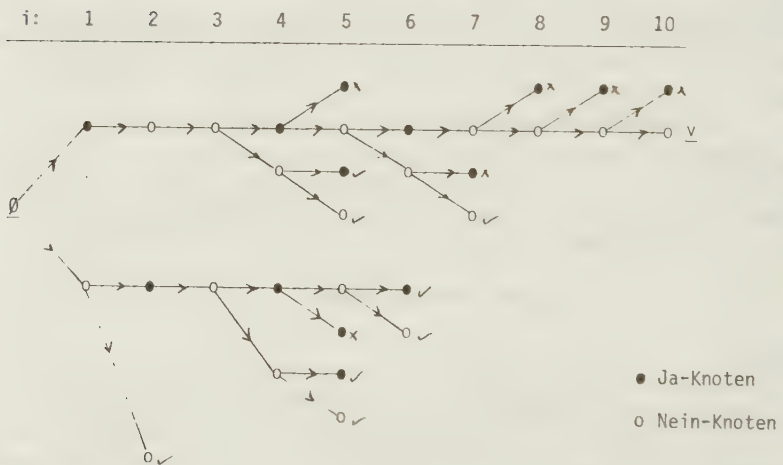
Bemerkung 3.2.6

Der Aufwand eines ε -approximierenden VP ist also kleiner als der des exakten VP, der sich auf die selbe VST bezieht. Diese Aufwandreduzierung wird erzielt, indem ein $100 \cdot \varepsilon$ -prozentiger Fehler des Ergebnisses in Kauf genommen wird.

Beispiel 3.2.7

Bild 3.2.1 zeigt den markierten Verzweigungsbaum, der sich nach einer Tour des 0,1-approximierenden VP ergibt. Die binäre Verzweigungsstruktur bezieht sich auf die in Bild 2.5.1 (Beispiel 2.5.17) angegebene K-Liste des Beispiels 2.3.45. Der Aufwand beträgt 33 Knoten (zum Vergleich: 48 Knoten beim exakten VP). Das 0,1-approximierende Ergebnis dieses speziellen Beispiels lautet:

$$((1,0,0,1,0,1,0,0,0,0), 165, 0,1).$$



$$\underline{v} = (1,0,0,1,0,1,0,0,0,0)$$

$$\omega(\underline{v}) = \hat{Z} = 165$$

Bild 3.2.1

3.3 Binärer Verzweigungsprozeß mit vorgegebener Rechenzeit

Der VP benötigt für seine Tour eine Laufphase, die der Ausführungsdauer des Prozesses im Rechensystem entspricht. Die Ausführungsdauer ändert sich von einem Problem zu einem anderen; sie ist vor der Lösung des Problems unbekannt. In der Praxis bekommt man vor Beginn des Rechenprozesses eine maximal zulässige Rechenzeit zugeteilt. Wenn diese Rechenzeit kleiner als die reale Zeit ist, die der Rechenprozeß benötigt, dann wird der Rechenprozeß abgebrochen, ohne eine Lösung liefern zu können. Die Frage lautet im allgemeinen: Gegeben sei das Problem P und die Rechenzeit T. Gesucht wird die beste Lösung des Problems P, die innerhalb der Rechenzeit T gefunden werden kann. In diesem Abschnitt werden wir uns auf das Problem der Phase 2 konzentrieren. Es wird angenommen, daß vor dem Beginn der Phase 2 die K-Liste bereits vorliegt und daß eine Rechenzeit T dem VP zur Verfügung steht. Gesucht wird ein approximierendes Ergebnis mit dem kleinsten $\mathcal{E}$ (am besten $\mathcal{E} = 0$). Der Algorithmus "BEST" beruht auf den folgenden Überlegungen: Man versucht zunächst für einen Bruchteil der gegebenen Rechenzeit (z.B. $T' = T/2$) den exakten VP (genannt VP(0), wobei gilt: $\mathcal{E} = 0$, $AS = 0$) ausführen zu lassen. Wird die Tour des VP(0) innerhalb T' beendet, dann erhält man die exakte(n) Lösung(en) des Problems. Sonst wird ein $\mathcal{E}_1 = c > 0$ (z.B. $c = 0,05$) und ein $AS_1 > 0$ gewählt. Mit diesen Parametern wird die Tour des neuen VP (genannt VP(1)) gestartet. Wenn sich die Tour von VP(1) nicht innerhalb T'' (z.B. $T'' = T'/2$) beenden läßt, dann wird ein VP(2) mit $\mathcal{E}_2 > \mathcal{E}_1$, $AS_2 > AS_1$ gestartet usw. Die Iteration wird solange fortgesetzt, bis sich schließlich ein Index i ergibt, bei dem die Tour des VP(i) beendet werden kann.

Algorithmus 3.3.1

Name: "BEST"

Voraussetzung: Die K-Liste des Problems liegt vor.

Sei $VP = (VST, SU, \mathcal{E})$ wie in Def. 3.2.1 angegeben.

$VP(i) = (VST, SU(i), \mathcal{E}(i))$ bezeichnet den VP bei der i-ten Iteration. $US(i)$ bezeichnet die untere Schranke, die sich nach der Beendigung oder nach dem Abbruch der Tour des VP(i) ergibt. $AS(i)$ bezeichnet die Anfangsschranke für VP(i). $MB(i)$ bezeichnet die Menge aller markierten Blätter $\underline{v}$, für die gilt: $\omega(\underline{v}) = US(i)$. $(MB(i), US(i), \mathcal{E}(i))$ bezeichnet das Ergebnis von VP(i) nach einer Tour. $t(i)$ bezeichnet die Rechenzeit für VP(i). T ist die Rechenzeit für die Phase 2. c und $\bar{c}$ sind Konstante ($c \ll 1$, $T \gg 4\bar{c}$).

Startbedingung: $i := 0$.

Iteration:

1. $\mathcal{E}(i) := 0$; $AS(i) := 0$; $t(i) := 2^{-1} * T - \widetilde{c}$.
2. Starte die Tour von $VP(i)$.
3. Ist die Tour von $VP(i)$ innerhalb $t(i)$ beendet ?
 Falls ja, $E := (MB(i), US(i), \mathcal{E}(i))$ und Sprung nach 8.
 Falls nein, $E := US(i)$ und die Tour des $VP(i)$ wird abgebrochen.
4. $i := i + 1$.
5. $\mathcal{E}(i) := 2^{i-1} * c$.
 Wenn $\mathcal{E}(i) \geq 1$, Sprung nach 8.
6. $AS(i) := E * (1 - \mathcal{E}(i))$.
 $t(i) := 2^{-(i+1)} * T - \widetilde{c}$.
 Wenn $t(i) \leq \widetilde{c}$, Sprung nach 8.
7. Sprung nach 2.
8. Ende.

Ergebnis: $E = (MB(i), US(i), \mathcal{E}(i))$, wenn die Tour des $VP(i)$ beendet wird.

$E = US(i)$, wenn die Tour des $VP(i)$ abgebrochen wird.

Bemerkung 3.3.2

$c \ll 1$ ist eine positive Konstante, die üblicherweise nach Erfahrung bzw. nach der zu erzielenden Genauigkeit des Ergebnisses gewählt werden soll. c läßt sich auch von $VP(0)$ selbst bestimmen, vorausgesetzt, daß bei der Tour des $VP(0)$ mindestens zwei zulässige Blätter der VST markiert werden. Seien $US_1, \dots, US_k$ ($k \geq 2$ und $US_k = US(0)$) die unteren Schranken, die sich bei der Tour des $VP(0)$ nacheinander ergeben.

$$c = US_k^{-1} * (k - 1)^{-1} * \sum_{i=2}^k (US_i - US_{i-1}) .$$

c wird also vom Verhalten des $VP(0)$ bestimmt. Auf diese Weise kann man eine gewisse Adaptivität in den Algorithmus einführen.

Bemerkung 3.3.3

Die Zeit für eine Iteration wird hauptsächlich von der Ausführungszeit von Schritt 2 bestimmt. Bei der i -ten Iteration entspricht die Ausführungszeit von Schritt 2 der Laufphase von $VP(i)$. Die Gesamtausführungszeit für die übrigen Schritte ist im Vergleich zur Laufphase von $VP(i)$ vernachlässigbar. Sie kann sich jedoch bemerkbar machen, wenn die Anzahl der Iterationen sehr groß wird (z.B. wenn c eine sehr kleine Konstante ist). Diese Möglichkeit wird durch den Test $t(i) \leq \tau$ bei Schritt 6 berücksichtigt, wobei τ die Summe der Ausführungszeiten der Schritte 3,4,5 und 6 angibt.

Satz 3.3.4

Der Algorithmus BEST bricht nach höchstens $\text{Min}(\lceil 1 + \log_2 \frac{1}{c} \rceil, \lceil \log_2(\frac{T}{\tau}) - 2 \rceil)$ Iterationen ab. Der Abbruch geschieht innerhalb der vorgegebenen Rechenzeit T .

Beweis:

BEST bricht spätestens nach k Iterationen ab, wenn es kein $i \leq k$ gibt, sodaß die Tour des $VP(i)$ innerhalb $t(i)$ beendet wird. In diesem Falle wird k von einer der folgenden Bedingungen bestimmt:

$$\text{i) } \mathcal{E}(k) \geq 1 \quad (\text{Schritt 5}).$$

$$\text{ii) } t(k) \leq \tau \quad (\text{Schritt 6}).$$

i) $\mathcal{E}(k) \geq 1$:

$$2^{k-1} * c \geq 1.$$

$$k \geq 1 + \log_2 \frac{1}{c} \quad (c > 0)$$

ii) $t(k) \leq \tau$:

$$2^{-(k+1)} * T - \tau \leq \tau.$$

$$2^{(k+1)} \geq \frac{T}{2 * \tau}.$$

$$k \geq \log_2 \left(\frac{T}{\tau} \right) - 2. \quad \left(\frac{T}{\tau} > 4 \right)$$

Die Bedingung mit dem kleineren k wird zuerst erfüllt. BEST bricht also nach höchstens $k = \text{Min}(\lceil 1 + \log_2 c^{-1} \rceil, \lceil \log_2(T * \tau^{-1}) - 2 \rceil)$ Iterationen ab.

Die i -te Iteration benötigt eine Ausführungszeit von

$$q_i = t(i) + \hat{C} = 2^{-(i+1)} \cdot T.$$

Sei Q die Gesamtausführungszeit für k Iterationen.

$$Q = \sum_{i=0}^{k-1} q_i = T \cdot \sum_{i=0}^{k-1} 2^{-i}.$$

$$\frac{Q}{T} = \sum_{i=0}^{k-1} 2^{-i} < 1.$$

Der Algorithmus wird also innerhalb der vorgegebenen Rechenzeit T abgebrochen. $\square$

Bemerkung 3.3.5

BEST beruht auf der in /10/ vorgeschlagenen Methode zur Auffindung der besten approximierenden Lösung unter der Voraussetzung, daß die zur Verfügung stehende Rechenzeit beschränkt ist. Dabei wird die Anzahl der Iterationen von $\mathcal{E}$ allein abhängig gemacht. In BEST spielen neben $\mathcal{E}$ zwei zusätzliche Parameter, nämlich AS und $\hat{C}$, eine große Rolle zur Bestimmung des Zeitpunkts für den Abbruch des Algorithmus.

Bemerkung 3.3.6

Wenn statt T die gesamte Rechenzeit S zur optimalen Zerlegung eines vorliegenden Netzes angegeben ist, kann man folgendermaßen vorgehen: Man läßt zunächst den Rechenprozeß der Phase 1 solange ablaufen, bis die K -Liste vorliegt. Sei T_1 die Rechenzeit der Phase 1. Ist $S - T_1 \gg \hat{C}$, so kann $(S - T_1)$ als die zur Verfügung stehende Rechenzeit der Phase 2 betrachtet werden. Sonst muß man zusätzliche Rechenzeit anfordern. Die Rechenzeit der Phase 3 ist im allgemeinen sehr klein im Vergleich zu den Rechenzeiten der Phase 1 und der Phase 2. Das Angsche Rechenverfahren hat bezüglich der Einteilung der zur Verfügung stehenden Rechenzeit den Vorteil, daß die Rechenprozesse der einzelnen Phasen ausschließlich von Größen abhängig sind, die entweder das Problem charakterisieren (die Phase 1) oder das Ergebnis der vorangehenden Phase darstellen (die Phase 2 und die Phase 3). Der Rechenprozeß jeder einzelnen Phase kann deshalb beliebig oft und zu beliebigem Zeitpunkt wiederholt werden.

3.4 Annäherungsverfahren mit abgekürzter K-Liste

Die Länge der K-Liste bestimmt im wesentlichen die Rechenzeit und den Speicherplatzbedarf der Rechenprozesse in der ersten und der zweiten Phase des Angschen Rechenverfahrens. Man interessiert sich deshalb für die Frage, wie die Anzahl der Elemente einer sehr langen K-Liste zu reduzieren ist, damit eine möglichst gute approximierende Lösung des Problems mit vertretbarem Aufwand gefunden werden kann. In diesem Abschnitt werden wir zwei Annäherungsverfahren untersuchen, die auf der Reduzierung der Länge der K-Liste beruhen.

Definition 3.4.1

Sei $KL = (KF, R_k, \alpha'_k)$ eine K-Liste. Sei l die Länge von KL . Sei $KF = \bigcup_{j \in V} K_j$, wobei K_j der Kern der Indikatorfamilie F_j sein soll. Sei r_j der Vorstand von K_j . Sei $S \triangleq \sum_{i \in V} r_i$. $KL' = (KF', R_k, \alpha'_k)$ heißt genau dann m-abgerundete Liste von KL , wenn es ein $m \in \mathbb{N}_1$ gibt, sodaß folgendes gilt:

$$i) \quad KF' \triangleq \{(\underline{s}_i, w_i) \mid (\underline{s}_i, w_i) \in KF \wedge i \leq m\}.$$

$$ii) \quad \alpha'_k : KF' \xrightarrow{\text{bij}} \mathbb{N}_m, \text{ angegeben durch}$$

$$\alpha'_k((\underline{s}_i, w_i)) = \begin{cases} i, & \text{wenn } (\underline{s}_i, w_i) \in KF'. \\ \text{undefiniert,} & \text{wenn } (\underline{s}_i, w_i) \notin KF'. \end{cases}$$

Sei $(\underline{s}_{m+1}, w_{m+1}) \in K_p$, $p \in V$, dann heißt

$$\lambda \triangleq (w_{m+1} + \sum_{j=p+1}^n r_j) / S \quad (n = |V|)$$

die Abweichung bezüglich KL .

Satz 3.4.2

Sei $KL' = (KF', R_k, \alpha'_k)$ die m -abgerundete Liste von $KL = (KF, R_k, \alpha'_k)$.

Sei $VST = (P, A, KF, \omega, \vartheta, \varphi, \underline{\vartheta})$ die binäre Verzweigungsstruktur bezüglich KL . Sei $VST' = (P', A', KF', \omega', \vartheta', \varphi', \underline{\vartheta}')$ die binäre Verzweigungsstruktur bezüglich KL' . Sei $\hat{Z}$ bzw. $\hat{Z}'$ das Maximum von VST bzw. von VST' .

$$\hat{Z} \geq \hat{Z}' \geq \hat{Z} * (1 - \varepsilon).$$

Dabei ist $\varepsilon = \lambda * S / \hat{Z}$.

Beweis:

Es gilt nach Def. 3.4.1 $\hat{Z} \geq \hat{Z}'$.

Sei $\hat{Z} = \hat{Z}' + X$. Nach Def. 3.4.1 gilt: $X \leq w_{m+1} + \sum_{j=p+1}^n r_j$.

Daraus folgt $\hat{Z} \leq \hat{Z}' + \lambda * S$.

Es gilt also $\hat{Z}' \leq \hat{Z} \leq \hat{Z}' + \lambda * S$.

$$\hat{Z}' \geq \hat{Z} - \lambda * S = \hat{Z} * (1 - \lambda * S / \hat{Z})$$

$$\hat{Z} \geq \hat{Z}' \geq \hat{Z} * (1 - \varepsilon)$$

Dabei ist $\varepsilon = \lambda * S / \hat{Z}$. $\square$

Bemerkung 3.4.3

Die Approximation mit Hilfe der m-abgerundeten Liste ist besonders geeignet für große Probleme, deren K-Listen einen sehr langen, dünnen "Schwanz" besitzen. Für eine solche K-Liste gibt es ein $j \in \mathbb{N}_n$, $n = |V|$, sodaß $r_1 \gg r_j > r_{j+1} > \dots > r_n$, und $(n - j)$ größenordnungsmäßig gleich j ist. Das Problem 2.8.1 ist ein solches Beispiel. Bild E.1 (Anhang E) zeigt die 18-abgerundete Liste des Problems 2.8.1. Diese Liste enthält anstatt 37 nur 20 Elemente, d.h. die ursprüngliche K-Liste wird um fast die Hälfte abgekürzt. Bild E.2 (Anhang E) veranschaulicht den binären Verzweigungsprozeß, der sich auf VST' bezieht.

Das Ergebnis: $Z' = \omega(\underline{v}) = \omega(\underline{u}) = 85$.

$$BM(\underline{v}) = \{ \{27, 28, 29\}, \{31, 33\}, \{6, 7\}, \{8, 9\}, \{17, 19\}, \{23, 24\} \}.$$

$$BM(\underline{u}) = \{ \{27, 28, 29\}, \{31, 33\}, \{6, 7\}, \{8, 9\}, \{17, 18\}, \{23, 24\} \}.$$

$$\frac{\hat{Z} - \hat{Z}'}{\hat{Z}} = 7,6\% \leq \varepsilon = \frac{\lambda * S}{\hat{Z}} = 23\%.$$

Dabei ist $\hat{Z} = 92$ das Optimum des Problems 2.8.1 (Anhang D).

Definition 3.4.4

Sei $KL = (KF, R_k, \alpha_k)$ eine K-Liste der Länge 1. Sei $KF = \bigcup_{i \in V} K_i$.

Sei r_i der Vorstand von K_i für $\forall i \in V$.

$KL'' = (KF'', R_k, \alpha_k'')$ heißt genau dann δ -reduzierte Liste, wenn gilt:

i) $\delta \in [0,1)$.

ii) $KF'' = \bigcup_{i \in V} K_i''$, wobei

$$K_i'' = \{(\underline{s}, w(\underline{s})) \mid (\underline{s}, w(\underline{s})) \in K_i \wedge w(\underline{s}) \geq \delta \cdot r_i\}.$$

iii) Sei $|KF''| = p$.

$\alpha_k'' : KF'' \xrightarrow{\text{bij}} \mathbb{N}_p$, angegeben durch

$$\alpha_k''(\underline{s}, w(\underline{s})) \leq \alpha_k''(\underline{t}, w(\underline{t})) \iff (\underline{s}, w(\underline{s})) R_k (\underline{t}, w(\underline{t})).$$

$\alpha_k''(\underline{s}, w(\underline{s}))$ bzw. $\alpha_k''(\underline{t}, w(\underline{t}))$ heißt Listennummer von $(\underline{s}, w(\underline{s}))$ bzw. von $(\underline{t}, w(\underline{t}))$ in KL'' . p heißt Länge von KL'' .

Bezeichnung 3.4.5

Sei $\alpha_k''(\underline{s}, w(\underline{s})) = i \in \mathbb{N}_p$. Mit $(\underline{s}_i, w_i)$ wird der Blockindikator $(\underline{s}, w(\underline{s}))$ bezeichnet. D.h. jeder Blockindikator der Liste KL'' wird mit seiner Listennummer in KL'' indiziert.

Bemerkung 3.4.6

Es ist leider nicht gelungen, für das Maximum der binären Verzweigungsstruktur $VST'' = (P'', A'', KF'', \mathcal{P}'', \omega'', \varphi'', \mathcal{Z}'')$, die sich auf KL'' bezieht, einen unteren Grenzwert anzugeben. Der obere Grenzwert ist wiederum $\hat{Z}$, das Maximum der auf KL bezogenen VST. Der binäre Verzweigungsprozeß $VP'' = (VST'', SU'', 0)$ kann deshalb nur als eine Heuristik betrachtet werden. Für manche Probleme kann diese Heuristik zu einer sehr guten approximierenden Lösung führen. Das Problem 2.8.2 ist ein solches Beispiel, dessen 0,5-reduzierte Liste in Bild E.3 (Anhang E) angegeben wird. Bild E.4 (Anhang E) veranschaulicht den entsprechenden, binären Verzweigungsprozeß VP .

Das Ergebnis: $\hat{Z}'' = \omega(\underline{u}) = 494$.

$$BM(\underline{u}) = \{\{6,7,8,9\}, \{2,3,4,5\}, \{10,11,12,13\}\}.$$

$$\frac{\hat{Z} - \hat{Z}''}{\hat{Z}} < 0,3 \% .$$

Dabei ist $\hat{Z} = 495$ das Optimum des Problems 2.8.2 (Anhang D).

Es scheint, daß diese Heuristik besonders geeignet ist für diejenigen Probleme, deren K-Liste aus vielen Kernen K_i mit $|K_i| \gg 1$ besteht, bei denen $w(\underline{s}) \ll r_i$ für die meisten $(\underline{s}, w(\underline{s})) \in K_i$ gilt, die nicht Vorstand sind.

3.5 Probleme mit einer oberen und einer unteren Gewichtsgrenze

Es handelt sich hier um Probleme, bei denen die Bedingung $a_1 \geq \sum_{i \in b} g_i \geq a_2$, $a_1, a_2 \in \mathbb{N}$, erfüllt werden muß. Dabei bezeichnet $a_1, \sum_{i \in V} g_i > a_1$, die obere Gewichtsgrenze und $a_2 \geq 0$ die untere Gewichtsgrenze des Problems. Zwei Fälle lassen sich unterscheiden: $a_1 = a_2$ oder $a_1 > a_2$. In beiden Fällen ist b genau dann ein Block des Netzes $NG = (V, E, g, f)$, wenn $b \subseteq V$ und $a_1 \geq \sum_{i \in b} g_i \geq a_2$. Probleme dieses Typs werden mit $P' = (NG, a_1, a_2, Z)$ bezeichnet. Das Problem $P = (NG, a, Z)$ läßt sich als ein spezieller Fall von P' betrachten, in dem folgendes gilt: $a = a_1$ und $a_2 = 0$.

Lemma 3.5.1

Sei $NG = (V, E, g, f)$ ein Netz. Seien $a_1, a_2 \in \mathbb{N}$ die Gewichtsgrenzen. Seien

$$BF_h \equiv \left\{ b \mid b \subseteq V \wedge a_1 \geq \sum_{i \in b} g_i > 0 \wedge h = \text{Hauptknoten von } b \right\}.$$

$$BF'_h \equiv \left\{ b' \mid b' \subseteq V \wedge a_1 \geq \sum_{i \in b'} g_i \geq a_2 \wedge h = \text{Hauptknoten von } b' \right\}.$$

Dann gilt:

$$BF'_h \subseteq BF_h.$$

Beweis: Offensichtlich. $\square$

Lemma 3.5.2

Sei $WB_h \subseteq BF_h$ ($WB'_h \subseteq BF'_h$) die Menge aller wesentlichen Blöcke von BF_h (BF'_h).

$$WB'_h \subseteq WB_h.$$

Beweis:

WB'_h läßt sich folgendermaßen auffinden: man bildet zunächst WB_h mittels des Erweiterungsprozesses aus BF_h . Es werden dann alle Blöcke b' aus WB_h aufgenommen, die der Bedingung $a_1 \geq \sum_{i \in b'} g_i \geq a_2$ genügen. Man erhält die Menge

$$WB'_h = \left\{ b' \mid b' \in WB_h \wedge a_1 \geq \sum_{i \in b'} g_i \geq a_2 \right\}.$$

Daraus folgt $WB'_h \subseteq WB_h$. $\square$

Satz 3.5.3

Seien $P = (NG, a, Z)$, $P' = (NG, a_1, a_2, Z)$. Sei $a = a_1$. Sei F bzw. F' die Indikatormenge von P bzw. von P' . Sei KF bzw. KF' der Kern von F bzw. von F' .

$$KF' \subseteq KF.$$

Beweis:

$$KF' = \{(\underline{s}_i, w_i) \mid (\underline{s}_i, w_i) = \text{Blockindikator von } b' \in \bigcup_{h \in V} WB_h'\}.$$

$$KF = \{(\underline{s}_i, w_i) \mid (\underline{s}_i, w_i) = \text{Blockindikator von } b \in \bigcup_{h \in V} WB_h\}.$$

Die Aussage folgt dann aus Lemma 3.5.2. $\square$

Bemerkung 3.5.4

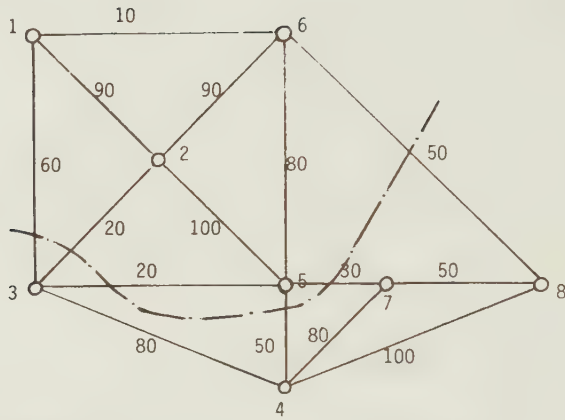
Es gilt also $|KF'| \leq |KF|$. $|KF'|$ nimmt im allgemeinen rapide ab, wenn sich die Werte von a_1 und a_2 nähern. Dementsprechend erhält man eine kürzere K -Liste, die zur Reduzierung des Aufwands des binären Verzweigungsprozesses führt. Der Aufwand läßt sich weiter reduzieren, wenn man von der so erhaltenen K -Liste ausgehend sich mit einem ε -approximierenden Ergebnis begnügt.

Beispiel 3.5.5

Dieses Beispiel wird der Arbeit von Christofides und Brooker /114/ entnommen. In dieser Arbeit wird ein Algorithmus angegeben, der eine optimale Zerlegung von Netzen unter den folgenden Voraussetzungen liefert:

- i) $g_i = 1$ für $\forall i \in V$.
- ii) $\sum_{i \in b} g_i = \frac{1}{2} n$.

Dabei ist b ein Block des zu zerlegenden Netzes $NG = (V, E, g, f)$ und $n = |V|$. Der Algorithmus teilt das Netz in zwei Blöcke auf, die jeweils die gleiche Anzahl von Knoten enthalten. Dies entspricht dem eingangs erwähnten Fall $a_1 = a_2 = \frac{1}{2} n$. Bild 3.5.1 zeigt das Netz und die Lösung des Problems. Die K -Liste und die schematische Darstellung des Verzweigungsprozesses findet sich in Anhang E (Bild E.5 und E.6).

Das NetzDie Lösung:

$$\hat{Z} = w(\underline{u}) = 680.$$

$$BM(\underline{u}) = \{\{1, 2, 5, 6\}, \{3, 4, 7, 8\}\}.$$

Bild 3.5.1

4. Die Optimierung des Prozeßverhaltens im Adreßraum

4.1. Speicher, Adreßraum, Speicherhierarchie und virtueller Speicher

Definition 4.1.1

Ein Speicherelement ist ein physikalischer Gegenstand (z.B. Flip-Flop, Ferritkern usw.), der zwei Zustände (z.B. Ein - Aus, Positiv - Negativ usw.) besitzt und sich zu jedem Zeitpunkt in genau einem der beiden Zustände befindet. Bezeichnen wir mit 1 den einen Zustand und mit 0 den anderen Zustand des Speicherelements e , dann heißt $\{0, 1\}$ die Zustandsmenge von e .

Sei $I = \{0, 1, \dots, l-1\}$. Eine Speicherzelle $c \triangleq (e_i)_{i \in I}$ ist eine endliche Folge jeweils untereinander (physikalisch) gleicher Speicherelemente, wobei e_i das i -te Speicherelement der Speicherzelle c bezeichnet.

$w \triangleq (x_i)_{i \in I}$ mit $x_i \in \{0, 1\}$ heißt Wort. x_i heißt das i -te Bit des Wortes und $|I| = l$ heißt Wortlänge. w bezeichnet den Inhalt der Speicherzelle c .

$v = \sum_{i \in I} 2^i x_i$ heißt momentaner Wert des Wortes w . Ein Wort heißt Befehlswort (kurz, Befehl), wenn es ein Element der Befehlsmenge des Rechensystems ist, sonst heißt es Datenwort.

Die Zeit, die zum Lesen des Inhaltes einer Speicherzelle bzw. zum Schreiben eines Wortes in die Speicherzelle benötigt wird, heißt Zugriffszeit.

Bemerkung 4.1.2

- i. Ein Wort der Länge l kann 2^l verschiedene Werte annehmen.
- ii. Wir betrachten ein idealisiertes Speicherelement, bei dem angenommen wird, daß die Umschaltzeiten, d.h. die Übergangszeit vom Zustand 0 zum Zustand 1 bzw. von 1 nach 0, null sind.
- iii. Es wird angenommen, daß in der Rechenanlage eine Hardware-Einrichtung existiert, die das Lesen eines Wortes aus der Speicherzelle und das Schreiben eines Wortes in die Speicherzelle durchführt. Wir bezeichnen diese Lese- und Schreiboperationen als Speicherzugriffe. Die Methode zur Durchführung der Speicherzugriffe wird Zugriffsverfahren genannt.

Definition 4.1.3

Ein Speicher ist ein Tripel $S \triangleq (C, A, \alpha)$ bestehend aus

C , einer nicht leeren Menge von Speicherzellen;

A , einer nicht leeren Menge, die Adreßmenge genannt wird; und

$\alpha: C \rightarrow A$, einer injektiven Abbildung, die Adressierungsfunktion genannt wird.

Sei $c \in C$. $\alpha(c) \in A$ heißt Adresse der Speicherzelle c in S .

S heißt homogen, wenn alle Speicherzellen $c \in C$ dieselbe Zugriffszeit, z.B. T , haben.

S heißt linear, wenn $A = \{0, 1, 2, 3, 4, \dots\}$, sonst heißt S nicht linear.

S heißt endlich, wenn C endlich und α eine bijektive Abbildung ist.

Sei S endlich. Dann heißt $K = |C|$ Speicherkapazität (kurz, Kapazität).

S heißt Speicher mit Blockeinteilung g.d. wenn

i. $\exists A_B \neq \emptyset$ und $\exists A_J \neq \emptyset$, sodaß

$$A \subseteq A_B \times A_J$$

ii. Es existiert eine Relation auf A :

$$\equiv \subseteq A \times A, \text{ angegeben durch}$$

$$\equiv \equiv \{(a_1, a_2) \mid (a_1, a_2) \in A \times A \wedge l(a_1) = l(a_2)\}$$

wobei für $\forall a = (x, y) \in A$, $l(a)$ die linke (erste) Komponente von a bezeichnet, d.h. $l(a) = x$.

Bemerkung 4.1.4

Bei einem nicht homogenen Speicher wird unter der Zugriffszeit des Speichers die mittlere Zugriffszeit verstanden.

Satz 4.1.5

Sei $S = (C, A, \alpha)$ ein endlicher Speicher der Kapazität K . Seien $c, c' \in C$.

i. $c = c' \iff \alpha(c) = \alpha(c')$

ii. $K = |C| = |A|$

Beweis: α ist für einen endlichen Speicher bijektiv. $\square$

Satz 4.1.6

$\equiv$ ist eine Äquivalenzrelation auf A .

Beweis: $\equiv$ ist reflexiv, symmetrisch und transitiv. $\square$

Satz 4.1.7

Sei $[a]_{\equiv}$ die Äquivalenzklasse von $a \in A$ bezüglich $\equiv (\{[a]_{\equiv} \equiv \{a' \mid a' \in A \wedge a \equiv a'\}\})$.

Sei $[A]_{\equiv}$ die Restklassenmenge von A bezüglich $\equiv (\{[A]_{\equiv} \equiv \{[a]_{\equiv} \mid a \in A\})$.

$[A]_{\equiv}$ ist eine Zerlegung von A .

Beweis: Der Beweis dieses Satzes ist in /264/ S.124 angegeben. $\square$

Bemerkung 4.1.8

Der Einfachheit wegen wird $[a]$ anstatt $[a]_{\equiv}$, $[A]$ anstatt $[A]_{\equiv}$ geschrieben.

$[A]$ wird oft als die durch $\equiv$ induzierte Zerlegung von A bezeichnet.

Definition 4.1.9

Sei $S = (C, A, \alpha)$ ein endlicher Speicher mit Blockeinteilung.

Ein Speicherblock von S ist ein Quadrupel $S(b) = (C', A', \alpha', b)$, wobei folgendes gilt:

$$C' \subseteq C$$

$$A' \in [A]$$

$$\alpha' : C' \xrightarrow{\text{bij}} A'$$

$$b = 1(a), \quad \text{wobei } [a] = A'$$

b heißt Blockname des Speicherblockes $S(b)$.

Sei $K = 2^n$, $n \in \mathbb{N}$, die Kapazität des Speichers S . Ein Speicherblock heißt Kachel genau dann wenn

$$A_B = \{0, 1, \dots, (2^p - 1)\}$$

$$A_J = \{0, 1, \dots, (2^q - 1)\}$$

für $\forall p, q \in \mathbb{N}_0$ und $p + q = n$.

Ist $S(b) = (C', A', \alpha', b)$ eine Kachel, dann heißt b Kachelnummer und $k = |C'|$ Kachelgröße.

Sei $S = (C, A, \alpha)$ mit $A \subseteq A_B \times A_J$ ein endlicher Speicher mit Blockeinteilung.

Satz 4.1.10 enthält unmittelbare Folgerungen der Definition 4.1.9.

Satz 4.1.10

Sei $S(b) = (C', A', \alpha', b)$ ein Speicherblock von S .

$$\text{i. } A' = \{(x, y) \mid x = b \wedge y \in A_J\}$$

$$\text{ii. } C' = \{\alpha'^{-1}(a) \mid a \in A'\}$$

Ist $S(b)$ eine Kachel, dann gilt:

$$\text{iii. } k = 2^q$$

$$\text{iv. } |[A]| = 2^p$$

Bemerkung 4.1.11

Eine Kachel ist also ein Speicherblock fester Größe ($k = 2^q$), dessen Name eine nicht negative ganze Zahl ist. Sowohl der Speicherblock als auch die Kachel sind als endliche, lineare Speicher zu betrachten. Die Kachel wird in modernen Rechensystemen als eine Verwaltungseinheit zum Zweck der Speicher-verwaltung eingeführt. Wir werden in diesem Kapitel ausschließlich Speicher mit Kacheln betrachten.

Bemerkung 4.1.12

Sei $S = (C, A, \alpha)$ ein endlicher Speicher mit Kacheln. Nach Satz 4.1.10 enthält S genau $m = 2^p$ Kacheln, die folgendermaßen angegeben werden:

$$S(0) = (C'_0, A'_0, \alpha'_0, 0)$$

$$S(1) = (C'_1, A'_1, \alpha'_1, 1)$$

$$\begin{array}{cc} - & - \\ - & - \end{array}$$

$$S(m-1) = (C'_{m-1}, A'_{m-1}, \alpha'_{m-1}, m-1)$$

Die Menge $\{S(i) \mid i \in A_B\}$ läßt sich aus diesem Grunde als eine durch $\equiv$ induzierte Zerlegung von S betrachten.

Definition 4.1.13

Ein Adreßraum ist ein Tripel $AR = (W, A, \wp)$, wobei folgendes gilt:

W ist eine nicht leere Menge von Worten gleicher Länge (Wortmenge);

A ist eine nicht leere Adreßmenge; und

$\wp \subseteq W \times A$ ist eine Relation aus W in A , angegeben durch:

$$\forall a, w, w' ((w, a) \in \mathcal{S} \wedge (w', a) \in \mathcal{S}' \implies w = w')$$

$\mathcal{S}$ heit Datenmenge des AdreBraumes. $(w, a) \in \mathcal{S}$ heit Datenelement.

Sei $z = (w, a) \in \mathcal{S}$ ein Datenelement. Mit $l(z)$ bzw. $r(z)$ soll die linke bzw. die rechte Komponente des Paares (w, a) bezeichnet werden, d.h. $l(z) \hat{=} w$ und $r(z) \hat{=} a$.

AR heit linear, wenn $A = \{0, 1, 2, 3, 4, \dots\}$, sonst heit AR nicht linear.

AR heit endlich, wenn sowohl W als auch A endlich ist.

Ist AR endlich, dann heit $G = |A|$ AdreBraumgre.

AR heit segmentiert g.d.w. AR endlich ist und

$$i. \exists A_N \neq \emptyset \exists A_M \neq \emptyset, \text{ soda} \quad A \subseteq A_N \times A_M$$

$$A \subseteq A_N \times A_M$$

ii. Es gibt eine Relation auf A :

$$\sim \subseteq A \times A, \text{ angegeben durch}$$

$$\sim \hat{=} \{(a_1, a_2) \mid (a_1, a_2) \in A \times A \wedge l(a_1) = l(a_2)\}$$

Bemerkung 4.1.14

Sei $AR = (W, A, \mathcal{S})$ ein segmentierter AdreBraum. Folgende Aussagen ergeben sich aus der Definition 4.1.13:

a) $\sim$ ist eine quivalenzrelation auf A .

b) $[A]_{\sim}$, die Restklassenmenge von A bezglich $\sim$, ist eine durch $\sim$ induzierte Zerlegung von A .

c) $\mathcal{S}^{-1}$ ist eine Abbildung.

Definition 4.1.15

Sei $AR = (W, A, \mathcal{S})$ ein segmentierter AdreBraum.

Ein Segment von AR ist ein Quadrupel $SG(\sigma) = (W', A', \mathcal{S}', \sigma)$:

$$W' \subseteq W$$

$$A' \in [A]$$

$$\mathcal{S}' \subseteq W' \times A' \quad \text{ist eine Relation, fr die folgendes gilt:}$$

$$\forall a, w, w' ((w, a) \in \mathcal{S}' \wedge (w', a) \in \mathcal{S}' \implies w = w')$$

$$\sigma = l(a), \text{ wobei } [a]_{\sim} = A' \text{ die quivalenzklasse von } a \in A \text{ bezglich } \sim \text{ ist.}$$

$\sim$ heißt Segmentname, $d = |A'|$ heißt Segmentgröße.

$SG(\sim)$ heißt linear, wenn $A_M = \{0, 1, \dots, (d-1)\}$.

$SG(\sim)$ ist in Seiten eingeteilt g.d.w.

$$i \mid d = 2^t \text{ und } A_M = A_P \times A_I$$

$$\text{mit } A_P = \{0, 1, \dots, (2^u - 1)\},$$

$$A_I = \{0, 1, \dots, (2^v - 1)\},$$

wobei $t, u, v \in \mathbb{N}$ und $t = u + v$ gilt.

ii. Es gibt eine Relation auf A' :

$$\simeq \subseteq A' \times A', \text{ angegeben durch}$$

$$\simeq \equiv \{(a_1, a_2) \mid (a_1, a_2) \in A' \times A' \wedge l(a_1) = l(a_2) \wedge m(a_1) = m(a_2)\},$$

wobei für $\forall a = (x, y, z) \in A'$ folgendes gilt:

$$l(a) = x, \quad m(a) = y, \quad r(a) = z.$$

Bemerkung 4.1.16

Sei $SG(\sigma) = (W', A', \mathcal{G}', \sigma)$ ein Segment des Adrebraumes $AR = (W, A, \mathcal{G})$.

Folgende Aussagen ergeben sich aus der Definition 4.1.15:

$$a) A' = \{(x, y) \mid x = \sigma \wedge y \in A_M\}.$$

$$b) \mathcal{G}' \subseteq \mathcal{G}.$$

$$c) (\mathcal{G}')^{-1} \text{ ist eine Abbildung.}$$

Ist $SG(\sigma) = (W', A', \mathcal{G}', \sigma)$ ein Segment mit Seiteneinteilung, dann gilt:

$$d) \simeq \text{ ist eine Äquivalenzrelation auf } A'.$$

e) $[A']_{\simeq}$, die Restklassenmenge von A' bezüglich $\simeq$, bildet eine durch $\simeq$ induzierte Zerlegung von A' .

$$f) A' = \{(x, y, z) \mid x = \sigma \wedge y \in A_P \wedge z \in A_I\}.$$

Definition 4.1.17

Sei $SG(\sigma) = (W', A', \mathcal{G}', \sigma)$ ein Segment, das in Seiten eingeteilt ist.

Eine Seite des Segments $SG(\sigma)$ ist ein Quintupel $PG(\sigma, \pi) = (W'', A'', \mathcal{G}'', \sigma, \pi)$:

$$W'' \subseteq W', \quad A'' \in [A'],$$

$\mathcal{Y}'' \subseteq W'' \times A''$ ist eine Relation, für die folgendes gilt:

$$\forall a, w, w' ((w, a) \in \mathcal{Y}'' \wedge (w', a) \in \mathcal{Y}'' \implies w = w'),$$

$\pi = m(a)$, wobei $[a]_{\sim} = A''$ die Äquivalenzklasse von $a \in A'$ bezüglich $\sim$ ist.

π heißt Seitennummer. $g = |A''|$ heißt Seitengröße. Besteht AR aus einem einzigen Segment, dann bezeichnet π die Seite mit der Nummer π in AR.

Bemerkung 4.1.18

Folgende Aussagen ergeben sich aus der Definition 4.1.17:

- a) $A'' = \{(x, y, z) \mid x = \sigma \wedge y = \pi \wedge z \in A_1\}$.
- b) $g = |A''| = 2^V$.
- c) $|[A']| = 2^U$.
- d) $(\mathcal{Y}'')^{-1}$ ist eine Abbildung.

Bemerkung 4.1.19

Eine Seite ist an sich ein endlicher, linearer Adreßraum. Ein Segment ist ein endlicher Adreßraum, der entweder linear oder in Seiten eingeteilt sein kann. Sei AR ein Adreßraum, der mehrere Segmente enthält, d.h. $|A_N| > 1$. Besteht A_N aus Alphanumerischen Symbolen, dann wird AR an manchen Literaturstellen (z.B. /204/) als symbolisch segmentierter Adreßraum bezeichnet, um ihn vom linear segmentierten Adreßraum ($A_N = \{0, 1, \dots, (n-1)\}$, $n \in \mathbb{N}$) zu unterscheiden. Sind die Segmente in Seiten eingeteilt, dann gilt i.a. Seitengröße = Kachelgröße. Der Speicher wird als Speicher mit Seiteneinteilung bezeichnet. Im Gegensatz zu Kacheln, die reale Speicherblöcke sind, werden Seiten bzw. Segmente als virtuelle Speicherbereiche betrachtet. Dementsprechend wird eine Seite π als Inhalt der Kachel aufgefaßt, in der π abgespeichert ist.

Definition 4.1.20

Sei $S = (C, A, \alpha)$ ein Speicher. Die Charakteristik von S ist ein Quintupel $M \equiv (l, K, T, U, Q)$ bestehend aus

l = Wortlänge

K = Speicherkapazität

T = Zugriffszeit

$$U \in \{1n, p\}$$

$$Q = \begin{cases} \text{Seitengröße, falls } U = p \\ \wedge (\text{leer}) \quad , \text{ falls } U = 1n \end{cases}$$

wobei $1n$ = linear und p = Seiteneinteilung.

Bemerkung 4.1.21

$M = (l, K, T, U, Q)$ enthält die wesentlichsten Merkmale von $S = (C, A, \alpha)$, welche die Eigenschaften des Speichers als Betriebsmittel kennzeichnen.

Definition 4.1.22

Eine n -stufige Speicherhierarchie ist ein Netz $H_n \equiv (SM, TW, CH, DR)$:

$SM = \{S_1, S_2, \dots, S_n\}$ ist die Speichermenge ;

$TW \subseteq SM \times SM \setminus \Delta_{SM}$ ist die Datenwegmenge ;

$CH = (M_1, \dots, M_n)$ ist die Charakteristik von H_n ;

$DR : TW \rightarrow \mathbb{R}^+$ ist die Datenratefunktion.

Seien $i, j \in \mathbb{N}_n$. $M_i = (l_i, K_i, T_i, U_i, Q_i)$ bezeichnet die Charakteristik von S_i . Es gilt für $\forall S_i, S_j \in SM$ folgendes:

$$i. \quad i \leq j \Rightarrow T_i \leq T_j$$

$$ii. \quad i \leq j \Rightarrow K_i \leq K_j$$

$$iii. \quad i \leq j \Rightarrow l_i \leq l_j$$

S_i wird Speicher der i -ten Stufe in H_n genannt.

Sei $(S_i, S_j) \in TW$. Dann heißt (S_i, S_j) Datentransportweg (Kurz, Datenweg) von S_i nach S_j . $DR((S_i, S_j))$ heißt Datenrate von S_i nach S_j .

$K_H = \sum_{i \in \mathbb{N}_n} K_i$ heißt Speicherkapazität von H_n .

Bemerkung 4.1.23

In der Praxis wird S_1 als Primär- bzw. Arbeitsspeicher, S_2 als Sekundärspeicher und S_3 als Tertiärspeicher der Speicherhierarchie H_n bezeichnet. Im allgemeinen werden alle S_i , $i > 3$, Dateispeicher genannt.

Definition 4.1.24

Sei $H_n = (SM, TW, CH, DR)$ eine n -stufige Speicherhierarchie. Sei $SM = \{S_1, \dots, S_n\}$, wobei $S_i = (C_i, A_i, \alpha_i)$ für $i \in \mathbb{N}_n$. Sei $AR = (W, A, \vartheta)$ ein Adreßraum. Sei $\omega \subseteq \bigcup_{i \in \mathbb{N}_n} (\{i\} \times \alpha_i)$.

Ein virtueller Speicher auf H_n mit dem Adreßraum AR ist ein Tripel

$$VS = (AR, H_n, R), \text{ wobei}$$

$R \subseteq \mathcal{P} \times \omega$ eine Relation aus $\mathcal{P}$ in ω ist, für welche folgendes gilt:

$$\forall r_1, r_2, 0 \ (r_1, 0) \in R \wedge (r_2, 0) \in R \Rightarrow r_1 = r_2$$

R heißt Adressierungsrelation von VS .

Bemerkung 4.1.25

Aus der Definition 4.1.24 folgt, daß R^{-1} eine Abbildung ist. R ordnet der Speicherzelle $c \in C_i$ des Speichers $S_i \in SM$ mit der realen Adresse $\alpha_i(c)$ genau ein Datenelement (w, a) (d.h. das Wort w mit der virtuellen Adresse a) des Adreßraums zu. Dagegen können jedem Datenelement von AR mehrere Speicherzellen verschiedener Speicher der Speicherhierarchie zugeordnet werden. Diese Modellvorstellung stimmt mit der allgemeinen Praxis der Speicherverwaltung in modernen Rechensystemen überein. Als Beispiel betrachten wir ein Wort w in der Seite Π , dessen Adresse a ist. Π kann Inhalt einer Kachel im Sekundär- und / oder im Tertiärspeicher sein. Wenn Π benötigt wird, muß sie in den Primärspeicher gebracht werden; d.h. eine Kachel muß im Primärspeicher bereitgestellt werden, um Π aufzunehmen. Nach der Aufnahme bildet Π den Inhalt der entsprechenden Kachel. Dies bedeutet, daß es eine Speicherzelle im Primärspeicher, eine im Sekundärspeicher und / oder eine im Tertiärspeicher gibt, deren Inhalt w ist. $((w, a), (i, \alpha_i(c))) \in R$ läßt sich also folgendermaßen interpretieren: Das Wort w mit der virtuellen Adresse a findet sich in einer Speicherzelle c des Speichers S_i der Speicherhierarchie H_n , deren reale Adresse durch $\alpha_i(c)$ angegeben ist. Gelte z.B. $((w, a), (i, \alpha_i(c))) \in R$ und $((w, a), (j, \alpha_j(c'))) \in R$, wobei $i \neq j$, dann ist w sowohl in $c \in C_i$ als auch in $c' \in C_j$ enthalten. In einem modernen Rechensystem wird R mittels einer speziellen Hardware-Einrichtung mit dem dazugehörigen Software im Betriebssystem realisiert.

4.2 Programmschemata und speicherorientierte Programmgraphen

Ein Programm wird im allgemeinen als eine Folge von Anweisungen betrachtet, die gemäß der Syntax und der Semantik einer Programmiersprache abgefaßt werden. Diese Betrachtungsweise führt unausweichlich zu einer Definition des Programmes, welche sowohl von der Syntax als auch von der Semantik der entsprechenden Programmiersprache abhängig ist, sodaß sie sich nicht für die in diesem Kapitel beabsichtigten Untersuchungen eignet. Für die Untersuchung des Prozeßverhaltens im Adreßraum benötigt man vielmehr eine formale Beschreibung des Programmes, die insbesondere dessen Ablaufstruktur in den Vordergrund treten läßt und zugleich alle mit den Programmiersprachen zusammenhängenden Einzelheiten vernachlässigt, die keinen Einfluß auf die Struktur des Programmes haben. Die Idee für die formale Beschreibung der Ablaufstruktur beruht auf der Erkenntnis, daß sich die Menge der Anweisungen einer Programmiersprache in zwei Äquivalenzklassen unterteilen läßt: Eine Klasse von Anweisungen, deren Ausführung eine bedingte- bzw. unbedingte Verzweigung (Sprung) zur Folge hat; die zweite Klasse enthält alle Anweisungen ohne Verzweigung. Diese Modellvorstellung stimmt im Grunde mit den Programmschemata von Ianov /154/ überein, die bereits als Grundlage zur Untersuchung verschiedener Probleme der Programmverifikation, der globalen Programm-Optimierung erfolgreich eingesetzt wurden. Ein Programmschema wird als ein uninterpretiertes Programm, d.h. ein Programm ohne Semantik, betrachtet, das die Ablaufstruktur einer Klasse von Programmen beschreibt, bei der jedes Programm eine mögliche Interpretation des Programmschemas darstellt. In diesem Kapitel wird die von Luckham, Park und Paterson /171/ entwickelte formale Sprache als Modell der Programmiersprachen benutzt. Diese Sprache zeichnet sich dadurch aus, daß sie eine strikte Trennung zwischen Syntax und Semantik besitzt. Auf dieser Sprache beruhend wird der Begriff des Programmschemas und des Programmes gebildet. Für die graphentheoretische Darstellung des Ablaufs der Programme im Adreßraum wird der speicherorientierter Programmgraph eingeführt. Von diesem Programmgraphen wird das Prozeßmodell für die Untersuchung des Prozeßverhaltens im Adreßraum abgeleitet.

Definition 4.2.1

Sei $OZ = \{F, T, X\}$. Sei $HZ = \{(\,,\,), ', \cdot, \cdot', =\}$.
 $ZM = \mathbb{N} \cup OZ \cup HZ$ wird Zeichenmenge genannt.

Ein Operationssymbol bezeichnet entweder einen Berechnungsoperator, oder einen Verzweigungsoperator, oder einen Parameter (d.h. die Adresse einer Variablen oder einer Konstante), oder eine Wertzuweisung.

Seien $i, j \in \mathbb{N}$.

$F(i, j)$ bezeichnet einen Berechnungsoperator.

$T(i)$ bezeichnet einen Verzweigungsoperator.

$X(i)$ bezeichnet einen Parameter.

$:=$ bezeichnet eine Wertzuweisung.

Eine Anweisung ist entweder eine Berechnungsanweisung oder eine Verzweigungsanweisung. Die Anweisungen werden folgendermaßen angegeben:

1) Berechnungsanweisung

$$a. X(i) := F(k, j)(X(l_1), X(l_2), \dots, X(l_k))$$

wobei $a, i, j, k, l_1, l_2, \dots, l_k \in \mathbb{N}$

2) Verzweigungsanweisung

$$a. T(i)(X(j))u, v$$

wobei $a, i, j, u, v \in \mathbb{N}$

a heißt Adresse der Anweisung. u, v heißen Zieladressen (kurz, ZA).

Definition 4.2.2

Eine Programmiersprache, deren Zeichenmenge, Operationssymbole und Anweisungen gemäß der Definition 4.2.1 angegeben sind, heißt LPP-Sprache (Luckham, Park, Paterson - Sprache).

Bemerkung 4.2.3

Die "STOP"-Anweisung, die ursprünglich (/171/) in der LPP-Sprache enthalten war, wird hier durch eine Verzweigungsanweisung mit $u = v = a$ ersetzt.

Bezeichnung 4.2.4

Mit C wird die Menge aller Berechnungsoperatoren und mit B die Menge aller Verzweigungsoperatoren der LPP-Sprache bezeichnet. $A = B \cup C$ wird Operatormenge genannt. Sei α eine Anweisung, dann bezeichnet $\lambda(\alpha)$ die Adresse von α ; d.h. $\lambda(\alpha) = a$. Mit $O(\alpha)$ wird der Operator von α bezeichnet; d.h. $O(\alpha) \in C$ ($O(\alpha) \in B$), wenn α eine Berechnungsanweisung (Verzweigungsanweisung) ist.

Wir werden künftig für die Anweisungen der LPP-Sprache folgendes annehmen:

- i. A, B und C sind endliche, nicht leere Mengen.
- ii. Jede Anweisung wird durch genau ein Wort, dessen Adresse a ist, im Adreßraum dargestellt.

Definition 4.2.5

Sei $n \in \mathbb{N}$. Ein LPP-Programmschema (kurz, Programmschema) ist eine endliche, nicht leere Folge von Anweisungen

$$\Sigma = (\alpha_1, \alpha_2, \dots, \alpha_n, \alpha_{n+1}),$$

für die folgendes gilt:

- i. $\alpha_i, i \in \mathbb{N}_{n+1}$, ist eine Anweisung in $\Sigma \Rightarrow O(\alpha_i) \in A$
- ii. $\forall i \in \mathbb{N}_n: (j = i + 1 \Rightarrow \lambda(\alpha_j) = \lambda(\alpha_i) + 1)$
- iii. $\forall i \in \mathbb{N}_n: (O(\alpha_i) \in B \wedge u, v \text{ sind ZA von } \alpha_i \Rightarrow$
 $\exists k \exists l (k, l \in \mathbb{N}_{n+1} \wedge \lambda(\alpha_k) = u \wedge \lambda(\alpha_l) = v)$
- iv. $\forall i \in \mathbb{N}_n: (O(\alpha_i) \in B \Rightarrow \exists \alpha_j (j \in \mathbb{N}_n \wedge O(\alpha_j) \in B \wedge \lambda(\alpha_{i+1}) = \text{ZA von } \alpha_j))$
- v. $O(\alpha_{n+1}) \in B$
- vi. $u', v' \text{ sind ZA von } \alpha_{n+1} \Rightarrow u', v' \text{ sind Adressen von Anweisungen nicht in } \Sigma.$

Bezeichnung 4.2.6

Seien α_i, α_j Anweisungen in Σ . α_j wird von α_i angesprungen, in Zeichen $\alpha_i \rightsquigarrow \alpha_j$, wenn $O(\alpha_i) \in B$ und $\lambda(\alpha_j) = \text{ZA von } \alpha_i$. Falls ein beliebiges α_j zum Ausdruck gebracht werden soll, schreiben wir $\rightsquigarrow \alpha_j$, d.h. α_j wird angesprungen.

Definition 4.2.7

Sei $\underline{Z} = (\alpha_1, \dots, \alpha_{n+1})$ ein Programmschema. Sei XP die Menge aller Parameter in $\underline{Z}$. Sei D eine nicht leere Menge. Ein Programm bezüglich $\underline{Z}$ ist ein Paar $P = (\underline{Z}, I)$, wobei

$$I \equiv \{I_p, I_f, I_g\}$$

I besteht aus 3 Abbildungen:

$$I_p : XP \longrightarrow D,$$

$$I_f : C \longrightarrow \{f \mid f : D^k \longrightarrow D\}, \quad k \in \mathbb{N}.$$

$$I_g : B \longrightarrow \{g \mid g : D \longrightarrow \{0, 1\}\},$$

die folgendermaßen angegeben werden:

- i. $X(i) \in XP \implies I_p(X(i)) \in D$
- ii. $\exists k \in \mathbb{N}_n (O(\alpha_k) = F(k, j)) \implies I_f(F(k, j)) : D^k \longrightarrow D$
- iii. $\exists m \in \mathbb{N}_{n+1} (O(\alpha_m) = T(i)) \implies I_g(T(i)) : D \longrightarrow \{0, 1\}$

D heißt Wertbereich des Programmes. I heißt Interpretation von $\underline{Z}$ über D .

$I_p(X(i))$ wird Interpretation von $X(i)$, $I_f(F(k, j))$ Interpretation von $F(k, j)$, $I_g(T(i))$ Interpretation von $T(i)$ genannt.

Eine Anweisung wird interpretierte Anweisung genannt, wenn sowohl ihr Operator als auch ihre Parameter jeweils eine Interpretation erhalten.

Bemerkung 4.2.8

Ein Programm ist also ein Programmschema, dessen Anweisungen interpretiert sind. Dementsprechend wird das Programmschema als ein uninterpretiertes Programm betrachtet. Zur Menge der interpretierten Berechnungsanweisungen gehören alle Anweisungen zur Durchführung der arithmetischen Operationen, der logischen Verknüpfungen, der Datentransportoperationen und der Ein-Ausgabeoperationen. Die Ein-Ausgabeoperationen werden als Datentransportoperationen betrachtet, die den Datentransport zwischen dem Adreßraum des Programmes und dem Adreßraum des Betriebssystems durchführen. Eine interpretierte Verzweigungsanweisung wird als ein unbedingter Sprung, d.h. "goto", betrachtet, wenn ihre beiden Zieladressen miteinander identisch sind.

Beispiel 4.2.9

Das Programmschema sei folgendermaßen angegeben:

<u>Anweisungen</u>	<u>Bezeichnung</u>
10. $X(2) := F(1,1)(X(2))$	α_1
11. $T(1)(X(1))15,12$	α_2
12. $X(2) := F(2,2)(X(1),X(2))$	α_3
13. $X(1) := F(1,3)(X(1))$	α_4
14. $T(1)(X(1))11,11$	α_5
15. $T(1)(X(1))u,u$	α_6

wobei $u \notin \{10, -15\}$.

Seien $x = X(1)$, $y = X(2)$ und $D \subseteq \mathbb{N}_0$. Sei $I = \{I_p, I_f, I_g\}$ eine Interpretation von $\mathcal{Z}$ über D , die folgendermaßen angegeben wird:

$$I_p(x), I_p(y) \in D;$$

$$I_f(F(1,1))(y) = 1;$$

$$I_f(F(2,2))(x,y) = x * y;$$

$$I_f(F(1,3))(x) = x - 1;$$

$$I_g(T(1))(x) = \begin{cases} = 0, & \text{wenn } x = 0 \\ = 1, & \text{wenn } x > 0 \end{cases}$$

Mit I ergibt sich ein Programm P bezüglich $\mathcal{Z}$, das die Funktion $y = x!$ rechnet:

10 $y := 1$

11 Wenn $x = 0$, Sprung nach 15.

12 $y := x * y$

13 $x := x - 1$

14 Sprung nach 11.

15 Sprung nach u .

Definition 4.2.10

Sei $\underline{Z} = (\alpha_1, \dots, \alpha_{n+1})$ ein Programmschema. Sei $Q = \{\alpha_i \mid i \in \mathbb{N}_{n+1}\}$, d.h. die Menge aller Anweisungen in $\underline{Z}$. Sei $AR = (W, A, \varphi)$ ein Adreßraum.

AR ist genau dann Adreßraum des Programmschemas $\underline{Z}$ wenn gilt:

- i) Jede Anweisung in $\underline{Z}$ wird durch ein Wort $w \in W$ dargestellt, dessen Adresse $a \in A$ die Adresse der Anweisung ist, d.h. jede Anweisung wird durch ein Datenelement in AR dargestellt.
- ii) Es gibt eine Relation $ZR \subseteq Q \times \mathcal{P}$, die $\underline{Z}$ mit der Datenmenge von AR verknüpft. ZR wird angegeben durch:

$$ZR \equiv \{(\alpha_i, (w', a')) \mid (\alpha_i, (w', a')) \in Q \times \mathcal{P} \wedge \exists X(1)(X(1) = \text{Param. von } \alpha_i \wedge X(1) = a')\}.$$

ZR heißt Datenzugriffsrelation (kurz, Zugriffsrelation). w' heißt Datenwort, a' Zugriffsadresse von α_i .

$e = (w', a')$ heißt Datenelement von α_i in AR. Sei m die Wortlänge der Worte in W , dann heißt $v(e) = \{0, 1, \dots, 2^m - 1\}$ die Wertmenge von e .

Sei $DM(\alpha_i) = \{e \mid (\alpha_i, e) \in ZR\}$ die Datenmenge von α_i in AR.

$DM(\underline{Z}) = \bigcup_{i \in \mathbb{N}_{n+1}} DM(\alpha_i)$ heißt Datenmenge von $\underline{Z}$ in AR.

Sei P ein Programm bezüglich $\underline{Z}$, dann ist AR der Adreßraum von P .

Bemerkung 4.2.11

Nach der Bedingung i) der Definition 4.2.10 ist jede Anweisung an sich als ein Datenelement in AR zu betrachten. Für unseren Zweck wird jedoch davon ausgegangen, daß es keine α_j , $j \neq i$, in $\underline{Z}$ gibt, sodaß $(\alpha_i, (w', a')) \in ZR$ und $a' = \lambda(\alpha_j)$. D.h. es gibt keine Anweisung in $\underline{Z}$, die zugleich Datenelement einer anderen Anweisung in $\underline{Z}$ ist. Programme, welche die Eigenschaft der strikten Trennung von Anweisungen und Datenelementen besitzen, werden als "Reentrant Programs" (z.B. /221/) bezeichnet.

Für die beabsichtigten Untersuchungen ist es nicht nötig, Anweisungen und Datenworte der Programmschemata im einzelnen zu betrachten; sie sollen jeweils derart zusammengefaßt werden, daß grobere "Einheiten" als geeigneter Gegenstand der Betrachtung entstehen.

Definition 4.2.12

Sei $\Sigma = (\alpha_1, \dots, \alpha_{n+1})$ ein Programmschema. Sei $\alpha_j, j \in \mathbb{N}_{n+1}$, eine Anweisung in Σ . Ein linearer Programmabschnitt (kurz, LPA) von Σ ist eine Folge

$$L \equiv (\alpha_{l_1}, \dots, \alpha_{l_k}) \quad l_i \in \mathbb{N}_{n+1} \quad \text{für } \forall i \in \mathbb{N}_k,$$

für die folgendes gilt:

- i) Jede Komponente von L ist eine Anweisung von Σ .
- ii) $\forall i \forall j (i \neq k \wedge j \neq n+1 \wedge \alpha_{l_i} = \alpha_j \implies \alpha_{l_{i+1}} = \alpha_{j+1})$.
- iii) $\alpha_{l_1} = \alpha_j \implies \alpha_j$
- iv) $\alpha_{l_k} = \alpha_j \implies (j = n+1) \vee (j \in \mathbb{N}_n \wedge \exists \alpha_i (i \neq j \wedge \lambda(\alpha_i) = \lambda(\alpha_j) + 1 \wedge \alpha_i \circ \alpha_j))$

Beispiel 4.2.13

Das Programmschema des Beispiels 4.2.9 enthält folgende LPA:

$$(\alpha_1); (\alpha_2); (\alpha_3, \alpha_4, \alpha_5); (\alpha_6)$$

Satz 4.2.14

Sei $\Sigma = (\alpha_1, \dots, \alpha_{n+1})$ ein Programmschema. Sei $\alpha_j, j \in \mathbb{N}_{n+1}$, eine Anweisung in Σ .

$$0(\alpha_j) \in B \wedge \exists \alpha_i (\alpha_i \rightarrow \alpha_j) \implies (\alpha_j) \text{ ist ein LPA.}$$

Beweis:

Man prüft in einzelnen die Bedingungen der Definition 4.2.12.

Bed. i) ist trivialerweise erfüllt.

Bed. ii) ist erfüllt, da α_j die einzige Komponente von (α_j) ist.

Bed. iii) ist erfüllt, da α_j angesprungen wird und α_j die erste Komponente von (α_j) ist.

Bed. iv) Fall 1: $j = n+1$: α_j ist auch die letzte Komponente von (α_j) .

Fall 2: $j \neq n+1$: dann gibt es ein α_i mit $i = j+1$, d.h.

$\lambda(\alpha_i) = \lambda(\alpha_j) + 1$. Ist $\lambda(\alpha_i)$ die ZA von α_j , dann ist Bed. iv) erfüllt.

Gilt $\lambda(\alpha_i) \neq \text{ZA von } \alpha_j$, dann gibt es nach Def. 4.2.5 ein α_s , mit $\lambda(\alpha_i) = \text{ZA von } \alpha_s$. D.h. Bed. iv) ist erfüllt. $\square$

Das Verfahren zur Auffindung aller LPA eines vorgegebenen Programmschemas wird in zwei nacheinander durchzuführende Phasen aufgeteilt. Die erste Phase besteht aus einem Algorithmus zur Markierung aller Anweisungen des Programmschemas, die angesprungen werden. Die Markierung kann z.B. so erfolgen, daß die markierten Anweisungen die Marke "1" erhalten. In der zweiten Phase wird das derart markierte Programmschema in lineare Programmabschnitte zerlegt.

Algorithmus 4.2.15

Name: "Mark"

Voraussetzung: $\Sigma = (\alpha_1, \dots, \alpha_{n+1})$ ist ein Programmschema.

Startbedingung: Σ ist nicht markiert.

Iteration:

1. $i := 1$
2. markiere α_i .
3. wenn $0(\alpha_i) \in C$, Sprung nach 5
4. Markiere diejenigen $\alpha_k, k \in \mathbb{N}_{n+1}$, für die $\alpha_i \rightarrow \alpha_k$ gilt.
5. $i := i + 1$
6. Wenn $i \leq n+1$, Sprung nach 3
7. Halt.

Ergebnis: Σ ist markiert.

Bemerkung 4.2.16

Wenn α_k mehrmals angesprungen wird, wird es mehrmals markiert. Dies verursacht keinen Fehler, da bei jeder Markierung α_k immer dieselbe Marke erhält.

Satz 4.2.17

- i) "Mark" bricht nach genau $n+1$ Iterationen ab.
- ii) Nach Abbruch des Algorithmus werden α_1 und diejenigen α_k in Σ markiert, die angesprungen werden.
- iii) Sei l die Anzahl der Verzweigungsanweisungen, m die Anzahl der markierten Anweisungen in Σ .

$$1 \leq m \leq 2l - 1$$

Beweis:

- i) Eine Iteration enthält die Schritte 3-5-6 oder 3-4-5-6. Nach der $(n+1)$ -ten Iteration (Schritt 6) bricht der Algorithmus ab.
- ii) α_1 wird bei Schritt 2 markiert.
Schritt 4 wird ausgeführt, wenn α_i eine Verzweigungsanweisung ist.
Alle α_k , für die $\alpha_i \leadsto \alpha_k$ gilt, erhalten eine Markierung.
- iii) Jede Verzweigungsanweisung hat genau 2 Zieladressen, die miteinander identisch sein können (unbedingter Sprung). Verschiedene Verzweigungsanweisungen können auch dieselben Zieladressen besitzen. Unter den 1 Verzweigungsanweisungen hat α_{n+1} spezielle Zieladressen, die nicht Adressen der Anweisungen von $\leq$ sind. Es gibt also höchstens $2(1 - 1)$ beim Schritt 4 markierte Anweisungen. Da α_1 auf jeden Fall beim Schritt 2 markiert ist, ergibt sich die folgende Aussage:

$$m \leq 2(1 - 1) + 1 = 21 - 1$$

$m \geq 1$ folgt aus der Tatsache, daß mindestens α_1 markiert sein muß. $\square$

Beispiel 4.2.18

Im Programmschema des Beispiels 4.2.9 werden die Anweisungen $\alpha_1, \alpha_2, \alpha_3$ und α_6 markiert.

Bezeichnung 4.2.19

Mit $L(j)$, $j \in \mathbb{N}$, wird die j -te Folge von Anweisungen bezeichnet. Sei $L(j) = (\alpha_{j_1}, \alpha_{j_2}, \dots, \alpha_{j_k})$, dann bezeichnet $L(j) \circ \alpha_i$ die Folge $(\alpha_{j_1}, \alpha_{j_2}, \dots, \alpha_{j_k}, \alpha_i)$. $\emptyset$ bezeichnet die leere Folge $()$. Seien $L(m) = (\alpha_{m_1}, \dots, \alpha_{m_s})$, $L(n) = (\alpha_{n_1}, \dots, \alpha_{n_t})$. Dann bezeichnet $L(m) \circ L(n)$ die Folge $(\alpha_{m_1}, \dots, \alpha_{m_s}, \alpha_{n_1}, \dots, \alpha_{n_t})$.

Algorithmus 4.2.20

Name: "LPA"

Voraussetzung: $\leq = (\alpha_1, \dots, \alpha_{n+1})$ ist ein Programmschema.

Startbedingung: $\leq$ ist bereits markiert.

Iteration:

1. $i := 1; j := 0; L(0) := \emptyset$
2. Falls α_i nicht markiert ist, Sprung nach 5.
3. $j := j + 1$
4. $L(j) := \emptyset$
5. $L(j) := L(j) \circ \alpha_i$
6. $i := i + 1$
7. Wenn $i \leq n + 1$, Sprung nach 2
8. Halt

Ergebnis: $L(1), \dots, L(j)$ sind die linearen Programmabschnitte von Σ .

Satz 4.2.21

- a) "LPA" bricht nach genau $n+1$ Iterationen ab.
- b) Sei m die Anzahl der markierten Anweisungen in Σ , dann gilt nach Abbruch des Algorithmus $j = m$.
- c) Jede $L(j)$, $j \in \mathbb{N}_m$, ist ein LPA, dessen erste Komponente eine markierte Anweisung von Σ ist.
- d) $L(1) \circ L(2) \circ \dots \circ L(m) = \Sigma$.

Beweis:

- a) Eine Iteration enthält die Schritte 2-5-6-7 oder 2-3-4-5-6-7. Der Algorithmus bricht ab, wenn $i > n+1$ (Schritt 7).
- b) j wird bei jeder markierten Anweisung um 1 erhöht (Schritt 3).
- c) Sei α_i eine markierte Anweisung, dann werden bei der i -ten Iteration die Schritte 2-3-4-5-6-7 durchlaufen.

Die Schritte 3 und 4 bereiten eine neue Folge $L(j)$ vor, die zunächst leer ist. Jede markierte Anweisung, z.B. α_i , bildet mit Schritt 5 die erste Komponente von $L(j)$. Seien $\alpha_{i+1}, \dots, \alpha_{i+k}$ nicht markierte Anweisungen. Sei $\alpha_{i+k} \neq \alpha_{n+1}$, und α_{i+k+1} eine markierte Anweisung, dann bilden $\alpha_{i+1}, \dots, \alpha_{i+k}$ die zweite, dritte und $(k+1)$ -te Komponente von $L(j)$, wobei α_{i+k} die letzte Komponente von $L(j)$ ist. Gilt $\alpha_{i+k} = \alpha_{n+1}$, dann ist $L(j)$ die letzte Folge. Sonst wird mit α_{i+k+1} eine neue Folge

$L(j+1)$ vorbereitet, für die α_{i+k+1} die erste Komponente ist, und das Konstruktionsverfahren verläuft wie bei $L(j)$. Auf diese Weise sind die Bedingungen der Definition 4.2.12 erfüllt. D.h. $L(j)$, $j \in \mathbb{N}_m$, ist ein LPA, dessen erste Komponente eine markierte Anweisung ist.

d) Diese Aussage ergibt sich aus dem Konstruktionsverfahren von $L(j)$, das in c) erläutert wurde. $\square$

Sei $L(i)$ ein LPA. Der Einfachheit wegen wird künftig L_i statt $L(i)$ geschrieben. Mit $\alpha_{ij} \in L_i$ wird eine Anweisung in L_i bezeichnet, welche die i -te Komponente von L_i bildet.

Definition 4.2.22

Sei $DM(\underline{\Sigma})$ die Datenmenge des Programmschemas $\underline{\Sigma}$. Sei $PL = \{L_i \mid i \in \mathbb{N}_m\}$.

Die Relation $LR \subseteq PL \times DM(\underline{\Sigma})$ wird folgendermaßen angegeben:

$$LR \equiv \{(L_i, e) \mid (L_i, e) \in PL \times DM(\underline{\Sigma}) \wedge \exists \alpha_{ij} (\alpha_{ij} \in L_i \wedge (\alpha_{ij}, e) \in ZR)\}$$

Sei $y \in LR$. $l(y)$ bzw. $r(y)$ soll die linke bzw. die rechte Komponente von y bezeichnen. Die Menge

$$LM(e) \equiv \{l(y) \mid y \in LR \wedge r(y) = e\}$$

wird LPA-Menge des Datenelements e genannt. PL wird als LPA-Menge des Programmschemas bezeichnet.

Folgerung 4.2.23

$e \in DM(\underline{\Sigma}) \implies LM(e) \neq \emptyset$.

Definition 4.2.24

Die Relation $\Xi_{\lambda} \subseteq DM(\underline{\Sigma}) \times DM(\underline{\Sigma})$ wird folgendermaßen angegeben:

$$\Xi_{\lambda} \triangleq \{(e, e') \mid (e, e') \in DM(\underline{\Sigma}) \times DM(\underline{\Sigma}) \wedge LM(e) = LM(e')\}.$$

Bemerkung 4.2.25

Ξ_{λ} ist eine Äquivalenzrelation.

Definition 4.2.26

Sei $e \in DM(\Sigma)$. Die Äquivalenzklasse von e bezüglich $\equiv_\lambda$, $[e]_{\equiv_\lambda} = \{e' \mid (e, e') \in \equiv_\lambda\}$, wird Datenblock genannt.

Bemerkung 4.2.27

Die Restklassenmenge von $DM(\Sigma)$ bezüglich $\equiv_\lambda$, $[DM(\Sigma)]_{\equiv_\lambda} = \{[e]_{\equiv_\lambda} \mid e \in DM(\Sigma)\}$, ist eine Zerlegung von $DM(\Sigma)$.

Bezeichnung 4.2.28

Mit $DB = \{d_i \mid i \in \mathbb{N}_q\}$ wird die Restklassenmenge $[DM(\Sigma)]_{\equiv_\lambda}$ bezeichnet, wobei $|[DM(\Sigma)]_{\equiv_\lambda}| = q$. Jedes $d \in DB$ ist ein Datenblock. DB wird als Datenblockmenge bezeichnet.

Definition 4.2.29

Sei $d \in DB$ ein Datenblock. Sei $e \in d$ und $v(e)$ die Wertmenge von e . Sei $|d| = k$. $V(d) \triangleq v(e)^k = \underbrace{v(e) \times v(e) \times \dots \times v(e)}_{k \text{ mal}}$ heißt Wertmenge des Datenblocks d .

Definition 4.2.30

Sei DB die Datenblockmenge des Programmes P. Sei $D = \{d_1, \dots, d_s\}$, $s \in \mathbb{N}$, eine nicht leere Teilmenge von DB. Sei $V(D) = V(d_1) \times V(d_2) \times \dots \times V(d_s)$.

$$V \triangleq \bigcup_{D \in 2^{DB} \setminus \{\emptyset\}} V(D)$$

wird Wertmenge des Programms P genannt.

Folgerung 4.2.31

Sei l die Wortlänge der Datenworte.

- i) $\forall k \in \mathbb{N}: (|d| = k \implies |V(d)| = (2^l)^k = 2^{lk})$
 ii) $\forall s \in \mathbb{N}: (D = \{d_1, \dots, d_s\} \implies |V(D)| = |V(d_1)| * |V(d_2)| * \dots * |V(d_s)|)$

Lemma 4.2.32

$$\forall d \in DB \ (e, e' \in d \implies LM(e) = LM(e'))$$

Beweis:

Jedes d ist eine Äquivalenzklasse von e bzw. e' bezüglich $\equiv_{\lambda}$ (Def. 4.2.26).
Die Aussage ergibt sich dann aus Def. 4.2.24. $\square$

Beispiel 4.2.33

Wir betrachten das Programmschema Σ des Beispiel 4.2.9.

Den Parametern $X(1)$ und $X(2)$ entsprechend gibt es 2 Datenelemente von Σ :

$$e_1 = (w_1, X(1))$$

$$e_2 = (w_2, X(2))$$

Die LPA-Mengen sind:

$$LM(e_1) = \{(\alpha_2), (\alpha_3, \alpha_4, \alpha_5), (\alpha_6)\}$$

$$LM(e_2) = \{(\alpha_1), (\alpha_3, \alpha_4, \alpha_5)\}$$

Die Datenblockmenge von Σ ist:

$$DB = \{d_1, d_2\}$$

wobei

$$d_1 = \{e_1\}$$

$$d_2 = \{e_2\}$$

In den Definitionen 4.2.22, 4.2.24 und 4.2.26 sind Vorschriften enthalten, die zur Auffindung aller Datenblöcke eines gegebenen Programmschemas angewendet werden können. Zu diesem Zweck werden wir im folgenden einen Algorithmus angeben.

Algorithmus 4.2.34

Name: "Datenblock"

Voraussetzung: $\Sigma = (\alpha_1, \dots, \alpha_{n+1})$ ist ein Programmschema.

$PL = \{L(i) \mid i \in \mathbb{N}_m\}$ ist die LPA-Menge von Σ .

$DM(\Sigma) = \{e(j) \mid j \in \mathbb{N}_k\}$ ist die Datenmenge von Σ .

$LM(j)$ bezeichnet die LPA-Menge von $e(j)$.

LM bezeichnet die Menge aller $LM(j)$, $j \in \mathbb{N}_k$.

Startbedingung: $LM(j) = \emptyset$, $d(j) = \emptyset$ für $\forall j \in \mathbb{N}_k$

$LM = \emptyset$, $DB = \emptyset$.

Iteration:

1. $j := 1$
2. $i := 1$
3. $\exists \alpha (\alpha \in L(i) \wedge (\alpha, e(j)) \in ZR)$?
Falls nein, Sprung nach 5
4. $LM(j) := LM(j) \cup \{L(i)\}$
5. $i := i + 1$
6. Wenn $i \leq m$, Sprung nach 3
7. $i := 1$
8. Wenn $LM(j) = LM(i)$, Sprung nach 14
9. $i := i + 1$
10. Wenn $i \leq j$, Sprung nach 8
11. $LM := LM \cup \{LM(j)\}$
12. $d(j) := d(j) \cup \{e(j)\}$
13. $DB := DB \cup \{d(j)\}$, Sprung nach 15
14. $d(i) := d(i) \cup \{e(j)\}$
15. $j := j + 1$
16. Wenn $j \leq k$, Sprung nach 2
17. Halt

Ergebnis: $DB = \{d(j) \mid j \in \mathbb{N}_k \wedge d(j) \neq \emptyset\}$ ist die Datenblockmenge von Σ .

Bemerkung 4.2.35

Algorithmus 4.2.34 besteht aus 3 Iterationen:

Die Hauptiteration enthält die Schritte 2-3-4-5-6-7-8-9-10-11-12-13-14-15-16.

Die Iteration I enthält die Schritte 3-4-5-6.

Die Iteration II enthält die Schritte 8-9-10.

Satz 4.2.36

"Datenblock" bricht nach k Hauptiterationen ab mit dem Ergebnis, daß für $\forall j \in \mathbb{N}_k$ entweder $d(j) = \emptyset$ oder $d(j)$ ein Datenblock von Σ ist, d.h. $d(j) \in \text{DB}$.

Beweis:

Für jedes $j \in \mathbb{N}_k$ wird bei der Iteration I eine LPA-Menge $\text{LM}(j)$ des Datenelements $e(j)$ gebildet (Def. 4.2.22).

Bei der Iteration II wird die neu gebildete LPA-Menge, $\text{LM}(j)$, mit allen in LM enthaltenen LPA-Mengen verglichen (Schritt 8). Existiert ein $i \leq j$ sodaß $e(i) \in d(i)$ und $\text{LM}(i) = \text{LM}(j)$, dann gehören $e(i)$ und $e(j)$ zu demselben Datenblock (Lemma 4.2.32). Dabei wird $e(j)$ in die Menge $d(i)$ aufgenommen (Schritt 14). In diesem Falle bleibt $d(j) = \emptyset$, d.h. mit $e(j)$ wird keine neue Äquivalenzklasse bezüglich $\equiv_\lambda$ gebildet.

Gibt es für $e(j)$ keine $\text{LM}(i) \in \text{LM}$ sodaß $\text{LM}(i) = \text{LM}(j)$, dann ist $\text{LM}(j)$ eine neue LPA-Menge, die in LM aufgenommen wird (Schritt 11). $e(j)$ wird also das erste Element einer neuen Äquivalenzklasse bezüglich $\equiv_\lambda$, die den Datenblock $d(j)$ bildet (Schritt 12 und 13). Nach k Hauptiterationen (Schritt 16) befindet sich jedes $e(j)$, $j \in \mathbb{N}_k$, in genau einem der Datenblöcke von DB , wobei DB eine Zerlegung von $\text{DM}(\Sigma)$ ist (Bemerkung 4.2.27). $\square$

Bemerkung 4.2.37

Sei $|\text{DB}| = q$. Es gilt im allgemeinen $q \leq k = |\text{DM}(\Sigma)|$.

Definition 4.2.38

Sei PL die LPA-Menge eines Programmschemas. Seien $L_i, L_j \in PL$.

Sei $L_i = (\alpha_{i_1}, \dots, \alpha_{i_k})$ und sei $L_j = (\alpha_{j_1}, \dots, \alpha_{j_l})$. Ein Übergang

von L_i zu L_j , in Zeichen $L_i \xrightarrow{\alpha} L_j$, ist genau dann möglich wenn gilt:

$$(0(\alpha_{i_k}) \in C \wedge \lambda(\alpha_{j_1}) = \lambda(\alpha_{i_k}) + 1) \vee (\alpha_{i_k} \xrightarrow{\alpha} \alpha_{j_1})$$

Definition 4.2.39

Sei $\Sigma = (\alpha_1, \dots, \alpha_{n+1})$ ein Programmschema. Der Programmgraph von Σ ist ein Tripel:

$$PG = (PD, AE, MK)$$

Dabei ist

$PD \equiv PL \cup DB$ die Knotenmenge, für die folgendes gilt:

PL ist die LPA-Menge von Σ .

DB ist die Datenblockmenge von Σ .

$AE \equiv AUE$ die Kantenmenge, für die folgendes gilt:

$A \subseteq PL \times PL \setminus \Delta_{PL}$ ist die gerichtete Kantenmenge, angegeben

durch $A = \{(L, L') \mid (L, L') \in PL \times PL \setminus \Delta_{PL} \wedge L \xrightarrow{\alpha} L'\}$

$E \subseteq \binom{PD}{2}$ ist die Menge der ungerichteten Kanten, angegeben durch

$$E = \{\{L, d\} \mid L \in PL \wedge d \in DB \wedge \forall e \in d (\exists \alpha (\alpha \hat{=} L \Rightarrow (\alpha, e) \in ZR))\}$$

$MK \equiv \{L_a, L_f\} \subseteq PD$ die Menge der ausgezeichneten Knoten von PD, für die folgendes gilt:

$$L = L_a \iff L \in PL \wedge \alpha_1 \hat{=} L$$

$$L = L_f \iff L \in PL \wedge \alpha_{n+1} \hat{=} L$$

$L \in PL$ heißt Programmknoten. $d \in DB$ heißt Datenknoten. $(L, L') \in A$ heißt

Übergangskante. $(L, d) \in E$ heißt Datenzugriffskante - kurz, Zugriffskante -.

L_a (bzw. L_f) wird Anfangsknoten (bzw. Endknoten) des Programmgraphen genannt.

Folgerung 4.2.40

(PL, A) ist ein endlicher, gerichteter, schlichter Graph.

(PD, E) ist ein endlicher, ungerichteter, schlichter Graph.

Bemerkung 4.2.41

Der Programmgraph ist also ein endlicher, schlichter Graph mit zwei Knotentypen, dessen Übergangskanten gerichtet und dessen Zugriffskanten ungerichtet sind. Ist P das Programm, dessen Programmschema Σ ist, so wird PG als Programmgraph von P bezeichnet. Der Programmgraph beschreibt die statische Ablaufstruktur des entsprechenden Programmes, bei der die Übergangsrelation und die Zugriffsrelation zwischen den einzelnen LPA und Datenblöcken in den Vordergrund gestellt werden, ohne Einzelheiten über die darin enthaltenen Anweisungen und Datenelemente anzugeben.

Beispiel 4.2.42

Der Programmgraph des in Beispiel 4.2.8 angegebenen Programmschemas wird in Bild 4.2.1 dargestellt.

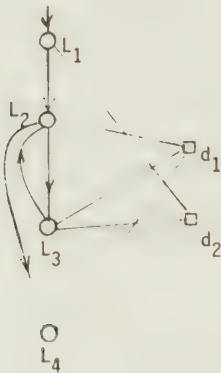


Bild 4.2.1

- Programmknoten
□ Datenknoten

$$PG = (PD, AE, MK)$$

$$PD = PL \cup DB$$

$$PL = \{L_1, L_2, L_3, L_4\}, \text{ wobei (Beispiel 4.2.13)}$$

$$L_1 = (\alpha_1); L_2 = (\alpha_2);$$

$$L_3 = (\alpha_3, \alpha_4, \alpha_5); L_4 = (\alpha_6).$$

$$DB = \{d_1, d_2\}, \text{ wobei (Beispiel 4.2.33)}$$

$$d_1 = \{e_1\}; d_2 = \{e_2\}.$$

$$AE = A \cup E$$

$$A = \{(L_1, L_2), (L_2, L_3), (L_3, L_2), (L_2, L_4)\}.$$

$$E = \{\{L_1, d_2\}, \{d_2, L_3\}, \{L_2, d_1\}, \{L_3, d_1\}, \{d_1, L_4\}\}.$$

$$MK = \{L_a, L_f\}, \text{ mit } L_a = L_1 \text{ und } L_f = L_4.$$

Definition 4.2.43

Sei $PG = (PD, AE, MK)$ ein Programmgraph. $PG' = (PD', AE', MK')$ heit zugeordneter Programmgraph von PG g.d.w. gilt:

$$PD' = PD$$

$$MK' = MK$$

$$AE' = A' \cup E', \text{ wobei}$$

$$E' = E$$

$$A' = \left\{ \{L, L'\} \mid \{L, L'\} \in \binom{PL}{2} \wedge (L \xrightarrow{e} L' \vee L' \xrightarrow{e} L) \right\}$$

Lemma 4.2.44

Sei PL die LPA-Menge und DB die Datenblockmenge eines Programmschemas.

$$\forall d \in DB \ (\exists L \ (L \in PL \implies \{d, L\} \in E))$$

Beweis:

Nach Definition 4.2.26 ist jedes $d \in DB$ die quivalenzklasse eines Datenelementes e bezglich $\equiv_A$, d.h. $d = [e]_{\equiv_A}$.

Nach Lemma 4.2.32 gibt es fr $\forall e \in d$ dieselbe $LM(e)$. Da $LM(e) \subseteq PL$ und $LM(e) \neq \emptyset$ (Folgerung 4.2.23), so folgt aus Def. 4.2.22, da es einen $L \in LM(e)$ gibt, soda $(L, e) \in LR$. D.h. $\exists \alpha \in L$ derart, da $(\alpha, e) \in ZR$. Daraus folgt, da $\{L, d\} \in E$. $\square$

Satz 4.2.45

PG' ist zusammenhngend $\iff (PL, A)$ ist zusammenhngend.

Beweis:

" $\implies$ " offensichtlich.

" $\impliedby$ " (PL, A) ist zusammenhngend $\implies (PL, A')$ ist zusammenhngend.

D.h. fr $L, L' \in PL$ gibt es einen Weg, der L mit L' verbindet. Fr jeden $d \in DB$ (bzw. $d' \in DB$) gibt es nach Lemma 4.2.44 einen $L \in PL$ (bzw. $L' \in PL$), soda $\{d, L\} \in E$ (bzw. $\{d', L'\} \in E$). Dies bedeutet, da fr $d, d' \in DB$ und $L, L' \in PL$ Wege existieren, die d mit L , d mit L' , d mit d' , L mit L' , d' mit L und d' mit L' verbinden. PG ist also zusammenhngend. $\square$

4.3 Dynamisches Prozeßverhalten im Adreßraum

Definition 4.3.1

Sei P ein Programm. Ein sequentieller Prozeß bezüglich P (kurz, Prozeß) ist ein Quintupel $\pi = (PG, V, \mathcal{P}, \omega, Z_M)$, wobei für $PG, V, \mathcal{P}, \omega, Z_M$ folgendes gilt:

$PG = (PD, AE, MK)$ ist der Programmgraph von P , für den gilt:

$$PD = PL \cup DB, \quad AE = A \cup E \quad \text{und} \quad MK = \{L_a, L_f\}.$$

V ist die Wertmenge von P .

$\mathcal{P} : Z \times V \longrightarrow Z$ heißt Zustandsübergangsfunktion (kurz, Übergangsfunktion)

$Z \subseteq PL \times 2^{DB}$ heißt Zustandsmenge von π in PG .

$\omega : Z \times V \longrightarrow V$ heißt Wertzuordnungsfunktion (kurz, Wertfunktion)

$Z_M \subseteq Z$ ist eine Menge von markierten Zuständen.

Z und Z_M werden folgendermaßen festgelegt:

- i) $Z \equiv \{(L, D) \mid (L, D) \in PL \times 2^{DB} \wedge D = \{d \mid \{L, d\} \in E\}\}$
- ii) $Z_M \equiv \{z_a, z_f\}$, wobei

$$z = z_a \iff z \in Z \wedge 1(z) = L_a$$

$$z = z_f \iff z \in Z \wedge 1(z) = L_f$$

z_a heißt Anfangszustand, z_f Endzustand von π .

Satz 4.3.2

$$|Z| = |PL|$$

Beweis:

Für jeden $L \in PL$ gibt es genau ein $D \in 2^{DB}$, sodaß $D = \{d \mid \{L, d\} \in E\}$. $\square$

Beispiel 4.3.3

Sei $PG = (PD, AE, MK)$ der in Beispiel 4.2.42 angegebene Programmgraph. Sei $\pi = (PG, V, \mathcal{P}, \omega, Z_M)$ ein Prozeß bezüglich des Programms P , dessen Programmgraph PG ist.

$$\begin{aligned}
Z &= \{(L_1, \{d_2\}), (L_2, \{d_1\}), (L_3, \{d_1, d_2\}), (L_4, \{d_1\})\} . \\
V &= \bigcup_{D \in 2^{DB} \setminus \{\emptyset\}} 2^{V(D)}, \text{ wobei } 2^{DB} \setminus \{\emptyset\} = \{\{d_1\}, \{d_2\}, \{d_1, d_2\}\} . \\
Z_M &= \{z_a, z_f\}, \text{ wobei } z_a = (L_1, \{d_2\}) \text{ und } z_f = (L_4, \{d_1\}) .
\end{aligned}$$

Definition 4.3.4

$\pi = (PG, V, \mathcal{S}, \omega, Z_M)$ ist genau dann laufbereit, wenn er sich im Endzustand befindet. In allen anderen Zuständen ist π laufend.

Bemerkung 4.3.5

"Laufend" und "laufbereit" können als globale Zustände des Prozesses aufgefaßt werden.

Definition 4.3.6

Sei $\pi = (PG, V, \mathcal{S}, \omega, Z_M)$ ein Prozeß bezüglich des Programms P. AR ist Adreßraum von π genau dann wenn AR Adreßraum von P ist.

Mit $\pi = (PG, V, \mathcal{S}, \omega, Z_M)$ wird ein Prozeßmodell definiert, das sich auf ein Programm bezieht, dessen Programmgraph PG die statische Ablaufstruktur des Prozesses angibt. Die Übergangsfunktion ist als eine Vorschrift anzusehen, die das dynamische Verhalten des Prozesses π im Adreßraum festlegt. Dabei kann man sich das so vorstellen, daß π das Programm ausführen muß, worauf sich π bezieht. Die Ausführung des Programms wird als Auftrag des Prozesses betrachtet. Jeder Prozeß hat also genau einen Auftrag auszuführen. Ist AR der Adreßraum von π , dann lassen sich die einzelnen Zustände von π als einzelne Bereiche im AR auffassen, wobei jeder Bereich als virtuelles Betriebsmittel des Prozesses zu betrachten ist. Der laufende Prozeß π fängt in einem bestimmten Bereich von AR an, der seinem Anfangszustand entspricht. π "wandert" dann von einem Bereich zu einem anderen. Dabei ändert sich sein Zustand ständig, bis er einen Bereich im AR erreicht hat, welcher seinem Endzustand entspricht; in diesem Bereich beendet π seinen Laufzustand und versetzt sich in den Zustand "Laufbereit". Vom Endzustand aus kann π wieder in den Anfangszustand versetzt werden, wenn er einen neuen Auftrag bekommt. Die Wertfunktion gibt bei jeder Zustandsänderung die Wertänderung des entsprechenden Datenblocks an.

In Kapitel 2 haben wir Prozesse definiert, die problemorientiert sind. Diese Prozesse sind z.B. der binäre Verzweigungsprozeß, der Erweiterungsprozeß usw.; jeder von ihnen hat die Aufgabe, ein gegebenes Problem zu lösen. Das in diesem Abschnitt entwickelte Prozeßmodell π kann als programmorientiert betrachtet werden, d.h. π bezieht sich auf ein Programm, das es ausführen muß. In π werden also die Struktur und andere Details des zu lösenden Problems nicht berücksichtigt. Die statische Ablaufstruktur des Prozesses wird beim problemorientierten Prozeßmodell durch die im Lösungsverfahren implizierte Ablaufstruktur festgelegt, im programmorientierten Prozeßmodell wird sie durch den Programmgraphen angegeben. Beide Modelltypen beruhen jedoch auf der gleichen Modellvorstellung, nämlich, daß unter einem Prozeß ein Tripel (α, β, γ) verstanden wird, wobei

α ein n-Tupel ist, der die statische Ablaufstruktur angibt;

β ein m-Tupel ist, der die Verhaltensregeln des Prozesses in α vorschreibt;

γ eine Menge ist, welche die speziellen Zustände des Prozesses kennzeichnet.

Gemäß dieser Modellvorstellung läßt sich sowohl das "Problem Lösen" als auch das "Programm Ausführen" als Auftrag betrachten. Aus diesem Grunde geben wir diesen beiden Modelltypen eine gemeinsame Bezeichnung: das auftragsorientierte Prozeßmodell. Mit Hilfe des programmorientierten Prozeßmodells läßt sich das dynamische Prozeßverhalten im Adreßraum als eine Folge von Aktivitäten des laufenden Prozesses betrachten, welche die (virtuellen) Betriebsmittelanforderungen des laufenden Prozesses beschreiben. Wir werden uns auf Eigenschaften des Prozesses konzentrieren, die mit den Betriebsmittelanforderungen eines laufenden Prozesses zusammenhängen. Andere Eigenschaften, welche für die Untersuchung des fehlerfreien Verhaltens von parallel kooperierenden Prozessen eine große Rolle spielen, können hier nicht berücksichtigt werden, ohne den Rahmen dieser Arbeit zu sprengen. Arbeiten über die fehlerfreien Eigenschaften von parallel kooperierenden Prozessen sind sehr zahlreich. Die ersten formalen Untersuchungen darüber finden sich z.B. in /242,250,252,258/. Manche der später entwickelten, nicht auftragsorientierten Prozeßmodelle haben auf den Ergebnissen dieser Arbeiten aufgebaut. Ein guter Überblick darüber findet sich in /245,248/.

Im folgenden sei $\pi = (PG, V, \mathcal{P}, \omega, Z_M)$ ein Prozeß mit der Zustandsmenge Z .

Sei $PG = (PD, AE, MK)$ mit $PD = PL \cup DB$, $AE = A \cup E$ und $MK = \{L_a, L_f\}$.

Seien $i, j \in \mathbb{N}_0$.

Definition 4.3.7

Seien $z, z' \in Z$ und es gelte $z \neq z'$. Ist $z = (L, D)$ zu einem bestimmten Zeitpunkt t der Zustand von π , so ist ein Übergang zum Zustand $z' = (L', D')$ genau dann möglich, wenn gilt:

$$(L, L') \in A \wedge \exists x \exists x' (x, x' \in 2^{V(D)}) \implies \mathcal{P}(z, x) = z' \wedge \omega(z, x) = x'$$

Der Übergang von z zu z' wird Ereignis zum Zeitpunkt t genannt. Mit $z \xrightarrow{E} z'$ wird das Auftreten des Ereignisses E bezeichnet, das π von z zu z' überführt.

Bezeichnung 4.3.8

Falls z bzw. z' beliebig sein soll, dann wird mit $o \xrightarrow{E} z'$ (bzw. $z \xrightarrow{E} o$) das Auftreten des Ereignisses E bezeichnet.

Definition 4.3.9

Sei $t_i \geq 0$ der Zeitpunkt, wobei $o \xrightarrow{E} z$. Sei $t_j > t_i$ der Zeitpunkt, wobei $z \xrightarrow{E} o$.

$$\mathcal{T} = t_j - t_i$$

heißt Verweilzeit des Prozesses im Zustand z , wenn es kein Ereignis E_k zum Zeitpunkt t_k mit $t_j > t_k > t_i$ gibt, das π in den Zustand $z' \neq z$ überführt. t_i (bzw. t_j) wird virtueller Zeitpunkt des Prozesses genannt.

Bemerkung 4.3.10

Es wird angenommen, daß die Zeit, welche π benötigt, um einen Übergang zu machen, im Vergleich zur Verweilzeit des Prozesses in einem Zustand vernachlässigbar ist.

Definition 4.3.11

Sei AR der Adreßraum von π . Sei Z die Zustandsmenge von π .

π geht einen Gang durch AR , wenn eine Folge von Ereignissen

$$E_0, \dots, E_i, E_{i+1}, \dots$$

existiert, für die folgendes gilt:

- i) $\forall i \in \mathbb{N}_0 (o \xrightarrow{E_i} z_i \implies z_i \in Z)$,
- ii) $o \xrightarrow{E_0} z_0 \implies z_0 = z_a$,
- iii) $\forall i \in \mathbb{N}_0 (z_i \xrightarrow{E_{i+1}} z_{i+1} \implies z_i \neq z_{i+1})$.

Wobei $z_i \in Z$ ($z_{i+1} \in Z$) den Zustand von π unmittelbar vor (nach) dem Eintreten des Ereignisses E_{i+1} bezeichnet.

Ein Gang ist genau dann eine Tour, wenn es ein E_n mit $n \geq 1$ gibt, sodaß

$$o \xrightarrow{E_n} z_n \wedge z_n = z_f.$$

Sei $E_0, \dots, E_i, E_{i+1}, \dots, E_n$ ein Folge von Ereignissen, die bei einer Tour des Prozesses π auftreten. Die Folge

$$z_0, \dots, z_i, z_{i+1}, \dots, z_n$$

wird Spur von π in AR genannt.

Definition 4.3.12

Seien $z, z' \in Z$. Die Übergangshäufigkeit von z zu z' ist eine Abbildung

$$h : Z \times Z \longrightarrow \mathbb{N}_0,$$

wobei $h_{(z,z')}$ die Anzahl der Ereignisse E mit $z \xrightarrow{E} z'$ angibt, die bei einer Tour des Prozesses π in AR auftreten.

Mit $h_{\{z,z'\}}$ wird die Übergangshäufigkeit zwischen z und z' bezeichnet, für die gilt:

$$h_{\{z,z'\}} = h_{(z,z')} + h_{(z',z)}$$

Definition 4.3.13

Seien $s, t \in \mathbb{N}$. Seien $L_s, L_t \in PL$ die Programmknotten von PG.

Ein Übergang von L_s zu L_t ist genau dann möglich, wenn

$$\exists z \exists z' \exists E (z \xrightarrow{E} z' \wedge L_s = l(z) \wedge L_t = l(z')).$$

Die Übergangshäufigkeit von L_s zu L_t ist eine Abbildung

$$h' : PL \times PL \longrightarrow \mathbb{N}_0,$$

wobei $h'_{(s,t)}$ die Anzahl der Übergänge von L_s zu L_t bei einer Tour des Prozesses angibt.

Mit $h'_{\{s,t\}}$ wird die Übergangshäufigkeit zwischen L_s und L_t bezeichnet, für die gilt:

$$h'_{\{s,t\}} = h'_{(s,t)} + h'_{(t,s)}$$

Folgerung 4.3.14

$$h'(s, t) = h(z, z')$$

$$h'_{\{s, t\}} = h_{\{z, z'\}}$$

Definition 4.3.15

Sei $z \in Z$. Die Aktivität des Prozesses π im Zustand $z \neq z_f$ ist ein Paar

$$q = (s, r) \in \mathbb{N} \times \mathbb{N}_0, \text{ wobei}$$

s = die Anzahl der ausgeführten Anweisungen in z ,

r = die Anzahl der Datenzugriffe in z .

r wird Zugriffshäufigkeit des Prozesses π in z genannt.

Sei $z = (L, D)$. Die Zugriffshäufigkeit auf $d \in D$, bezeichnet $r(d)$, ist die Anzahl der Datenzugriffe, welche π in z bezüglich des Datenblocks d gemacht hat.

Folgerung 4.3.16

$$r = \sum_{d \in D} r(d)$$

Definition 4.3.17

Seien $\circ \xrightarrow{E_i} z_i, z_i \xrightarrow{E_{i+1}} \circ$. Eine Aktivitätsphase des Prozesses π im Zeitintervall (t_i, t_{i+1}) ist ein Quadrupel

$$Q_i = (z_i, s_i, r_i, \tau_i) \in Z \times \mathbb{N} \times \mathbb{N}_0 \times \mathbb{R}^+,$$

wobei folgendes gilt:

- i) $z_i \in Z \wedge z_i \neq z_f$
- ii) (s_i, r_i) ist die Aktivität des Prozesses π in z_i .
- iii) τ_i ist die Verweilzeit des Prozesses π in z_i .

Gilt $z_i = z_f$, dann ist Q_i undefiniert.

Definition 4.3.18

Das dynamische Prozeßverhalten im Adreßraum ist eine Folge

$$Q_0, Q_1, \dots, Q_i, Q_{i+1}, \dots, Q_{n-1},$$

wobei für $\forall i \in \{0, 1, \dots, (n-1)\}$ $Q_i = (z_i, s_i, r_i, \tau_i)$ die Aktivitätsphasen des laufenden Prozesses sind und

$$z_0, \dots, z_i, z_{i+1}, \dots, z_n$$

seine Spur im Adreßraum ist.

$T = \sum_{i=0}^{n-1} \tau_i$ heißt Tourzeit des Prozesses.

Bemerkung 4.3.19

Die Tourzeit ist also die Zeit, die der laufende Prozeß benötigt, um das entsprechende Programm auszuführen. Die Aktivität des Prozesses im Adreßraum bei einer Tour läßt sich durch folgende 3 Parameter charakterisieren:

$S = \sum_{i=0}^{n-1} s_i$ ist die Gesamtzahl der ausgeführten Anweisungen,

$R = \sum_{i=0}^{n-1} r_i$ ist die Gesamtzahl der Datenzugriffe,

$H = \sum_{z, z' \in Z} h_{(z, z')}$ ist die Gesamtzahl der Zustandsübergänge.

Für den Rest dieses Abschnitts werden wir das dynamische Prozeßverhalten in einem nicht linearen Adreßraum betrachten. Dabei werden wir uns auf die Optimierung des Prozeßverhaltens in einem segmentierten Adreßraum mit Seiteneinteilung konzentrieren. Es hat sich herausgestellt, daß dieses Problem im Grunde mit der optimalen Zerlegung des speicherorientierten Programmgraphen identisch ist.

Definition 4.3.20

Sei $PG = (PD, AE, MK)$ ein Programmgraph. Sei $PG' = (PD', AE', MK')$ der zugeordnete Programmgraph von PG . Ein speicherorientierter Programmgraph ist ein Quadrupel $G_M = (K_M, E_M, g, f)$, wobei für K_M , E_M , g und f folgendes gilt:

$$K_M = PD'$$

$$E_M = AE'$$

$g : K_M \rightarrow \mathbb{N}$ ist eine Knotengewichtung, angegeben durch:

$$g(k) = n \iff \exists L(k = L \wedge |L| = n) \vee \exists d(k = d \wedge |d| = n).$$

$f : E_M \rightarrow \mathbb{N}$ ist eine Kantengewichtung, für die gilt:

$$f(\{x, y\}) = m \iff \exists L \exists L' \exists d((x = L \wedge y = L' \wedge m = h'(\{L, L'\}) \vee (x = L \wedge y = d \wedge m = r(d))).$$

$g(k)$ ist das Knotengewicht des Knotens k , $f(\{x, y\})$ das Kantengewicht der Kante $\{x, y\}$.

Satz 4.3.21

$G_M = (K_M, E_M, g, f)$ ist ein Netz $\iff PG'$ ist zusammenhängend.

Beweis:

Die Aussage ergibt sich aus der Definition 4.3.20. $\square$

Definition 4.3.22

Sei AR ein Adreßraum mit Seiteneinteilung. Sei $S = \{s_1, \dots, s_m\}$ die Menge aller Seiten im AR. Eine Anweisung α (ein Datenelement e) in der Seite s sei mit $\alpha \in s$ ($e \in s$) bezeichnet.

π ist zu einem gegebenen Zeitpunkt genau dann im Adreßraum der Seite $s_i \in S$ (kurz, π ist in s_i), wenn π eine Anweisung in s_i ausführt.

Ist π in s_i und zum gegebenen Zeitpunkt führt π eine Anweisung $\alpha_i \in s_i$ aus, dann wird eine neue Seite $s_j \neq s_i$ g.d. von π gefordert, wenn gilt:

$$\exists \alpha_j \in s_j (\alpha_i \rightsquigarrow \alpha_j \vee \lambda(\alpha_j) = \lambda(\alpha_i) + 1) \vee \exists e_j \in s_j ((\alpha_i, e_j) \in ZR)$$

Die Forderung einer neuen Seite wird Seitenwechsel genannt. Mit $s_i \xleftrightarrow{\pi} s_j$ wird die Möglichkeit eines Seitenwechsels zwischen s_i und s_j bezeichnet.

Bemerkung 4.3.23

Ein Segmentwechsel läßt sich auf gleiche Weise definieren. Seien $\tilde{\sigma}, \tilde{\sigma}'$ Segmente. Mit $\tilde{\sigma} \xleftrightarrow{\pi} \tilde{\sigma}'$ wird die Möglichkeit eines Segmentwechsels zwischen $\tilde{\sigma}$ und $\tilde{\sigma}'$ bezeichnet.

Definition 4.3.24

Sei AR der Adreßraum des Prozesses π . Sei $S = \{s_1, \dots, s_m\}$ die Menge aller Seiten in AR.

Ein Programmgraph mit Verwaltungseinheiten ist ein Quadrupel

$$G_S = (K', E', g', f')$$

wobei für K' , E' , g' und f' folgendes gilt:

$K' = S$ ist die Knotenmenge.

$E' = \{\{x, y\} \mid \{x, y\} \in \binom{S}{2} \wedge x \xleftrightarrow{\pi} y\}$ ist die Kantenmenge.

$g' : K' \rightarrow \mathbb{N}$ ist die Knotengewichtung, angegeben durch

$g'(k') = \text{Seitengröße}$ für $\forall k' \in K'$

$f' : E' \rightarrow \mathbb{N}$ ist die Kantengewichtung, wobei

$f'(\{x, y\}) = \text{Anzahl der Seitenwechsel zwischen } x \text{ und } y$
für $\forall \{x, y\} \in E'$.

Definition 4.3.25

Sei AR der Adreßraum des Prozesses π . Sei $\Sigma = \{\sigma_1, \dots, \sigma_l\}$ die Menge aller Segmente im AR.

Ein Programmgraph mit Transporteinheiten ist ein Quadrupel

$$G_\Sigma = (K'', E'', g'', f'')$$

wobei für K'' , E'' , g'' und f'' folgendes gilt:

$K'' = \Sigma$ ist die Knotenmenge.

$E'' = \{\{x, y\} \mid \{x, y\} \in \binom{K''}{2} \wedge x \xleftrightarrow{\pi} y\}$ ist die Kantenmenge.

$g'' : K'' \rightarrow \mathbb{N}$ ist die Knotengewichtung, angegeben durch

$g''(k'') = \text{Segmentgröße}$ für $\forall k'' \in K''$,

$f'' : E'' \rightarrow \mathbb{N}$ ist die Kantengewichtung, wobei

$f''(\{x, y\}) = \text{Anzahl der Segmentwechsel zwischen } x \text{ und } y$
für $\forall \{x, y\} \in E''$.

Definition 4.3.26

Sei AR ein Adreßraum mit Seiteneinteilung (bzw. ein segmentierter Adreßraum). Das dynamische Prozeßverhalten in AR ist genau dann optimiert, wenn die Gesamtzahl der Seitenwechsel (bzw. Segmentwechsel) bei einer Tour des Prozesses minimiert wird.

Satz 4.3.27

Das dynamische Prozeßverhalten in einem Adreßraum mit Seiteneinteilung ist genau dann optimiert, wenn G_S das Ergebnis der optimalen Zerlegung von G_M ist.

Beweis:

Das dynamische Prozeßverhalten im Adreßraum ist optimiert

$\Leftrightarrow$ Die Gesamtzahl der Seitenwechsel bei einer Tour des Prozesses ist minimal (Def. 4.3.26)

$\Leftrightarrow \sum_{\{x,y\} \in E'} f'(\{x,y\})$ ist minimal (Def. 4.3.24)

$\Leftrightarrow G_S$ ist eine optimale Zerlegung von G_M . $\square$ (Satz 2.1.25)

Satz 4.3.28

Das dynamische Prozeßverhalten in einem segmentierten Adreßraum mit Seiteneinteilung ist genau dann optimiert, wenn $G_{\mathcal{P}}$ die optimale Zerlegung von G_S , und G_S die optimale Zerlegung von G_M ist.

Beweis:

Es ist zu beweisen:

- i) Das dynamische Prozeßverhalten im Adreßraum mit Seiteneinteilung ist g.d. optimiert, wenn G_S die optimale Zerlegung von G_M ist.
- ii) Das dynamische Prozeßverhalten in einem segmentierten Adreßraum ist g.d. optimiert, wenn $G_{\mathcal{P}}$ die optimale Zerlegung von G_S ist.
- i) ist identisch mit dem Satz 4.3.27, der bereits bewiesen wurde. Der Beweis für ii) verläuft genau so wie der Beweis für Satz 4.3.27. $\square$

4.4 Ein Konzept zur Optimierung des Prozeßverhaltens im Adreßraum

Auf Satz 4.3.28 beruhend wird in diesem Abschnitt ein Konzept zur Optimierung des Prozeßverhaltens im Adreßraum angegeben. Das Konzept besteht darin, die folgenden 3 Arbeitsgänge nacheinander durchzuführen:

- i) Der speicherorientierte Programmgraph G_M des Programms, auf das sich der Prozeß bezieht, wird aufgefunden.
- ii) G_M wird optimal zerlegt. Das Ergebnis ist ein Graph G_S mit Verwaltungseinheiten, bei dem jeder Knoten eine Seite darstellt, in der die entsprechenden Programmknoten (LPA) und Datenknoten (Datenblöcke) von G_M zusammengefaßt werden. Dadurch wird die Gesamtzahl der Seitenwechsel bei einer Tour des Prozesses minimiert.
- iii) G_S wird optimal zerlegt. Das Ergebnis ist ein Graph G_σ mit Transporteinheiten, bei dem jeder Knoten ein Segment darstellt, in dem die entsprechenden Seiten des Adreßraumes zusammengefaßt werden. Dadurch wird die Gesamtzahl der Segmentwechsel bei einer Tour des Prozesses minimiert.

Die Arbeitsgänge ii) und iii) enthalten ein gemeinsames Problem, das mittels des Angschen Verfahrens zur optimalen Zerlegung von Netzen gelöst werden kann. Die Durchführung der Arbeitsgänge ii) und iii) besteht also darin, die in Kapitel 2 und 3 entwickelten Prozesse und Algorithmen zu implementieren. Um den speicherorientierten Programmgraphen im Arbeitsgang i) aufzufinden, müssen Messungen durchgeführt werden. Solche Messungen lassen sich folgendermaßen klassifizieren:

1. Statische Messungen
2. Dynamische Messungen

4.4.1 Statische Messungen

Unter einer statischen Messung wird ein Meßverfahren verstanden, bei dem das zu messende Programm (Zielfprogramm genannt) als Daten betrachtet wird, die gemäß vorgegebenen Vorschriften analysiert und klassifiziert werden. Die in Abschnitt 4.2 angegebenen Algorithmen zur Auffindung aller LPA und Datenblöcke eines Programmschemas sind Beispiele solcher Vorschriften. Die von einem Compiler durchzuführende Syntax-Analyse /144/ läßt sich als ein spezielles statisches Meßverfahren auffassen. An verschiedenen Literaturstellen wird die statische Messung als "automatische Programm-Analyse"

/133,137,140,141,142/ bezeichnet; sie spielt bei den Untersuchungen der globalen Programm-Optimierung /143 - 160/, der Parallelität in Programmen /130,131/, der Software-Testprobleme /138/ und der Sprachorientierten Programmstatistik /134,135,136,139/ eine wichtige Rolle.

Für den speicherorientierten Programmgraphen $G_M = (K_M, E_M, g, f)$ lassen sich die Parameter K_M , E_M und g mittels der statischen Messung auffinden. Dazu wird ein Meßprogramm benötigt, welches das Zielprogramm analysiert. Ist das Zielprogramm in einer problemorientierten Sprache abgefaßt, so muß das Meßprogramm der Sprache entsprechend entwickelt werden. Liegt das Zielprogramm in einer maschinenunabhängigen, standardisierten internen Sprache vor, so läßt sich das Meßprogramm gezielt für diese interne Sprache entwickeln. Viele moderne Compiler /144/ bewältigen ihre Aufgaben in 2 nacheinander durchzuführenden Arbeitsphasen: In der ersten Phase wird das zu kompilierende Programm in einen maschinenunabhängigen internen Text übersetzt; das in Maschinencode abgefaßte Programm wird dann in der zweiten Phase erstellt. Diese Arbeitsteilung ermöglicht nicht nur große Flexibilität und kleineren Aufwand beim Kompilieren, sondern auch die Durchführung der globalen Programm-Optimierung für das im internen Text vorliegende Zielprogramm. Eine der wichtigsten Voraussetzungen für die globale Programm-Optimierung ist die Auffindung des Ablaufsgraphen ("Flow Graph" /152,153, 159,160/). Da ein Ablaufsgraph als ein Teilgraph des in Abschnitt 4.2 definierten Programmgraphen betrachtet werden kann, lassen sich die Ablaufsgraphen und die speicherorientierten Programmgraphen gleichzeitig auffinden. Dies bedeutet, daß die Optimierung des Prozeßverhaltens im Adreßraum und die globale Programm-Optimierung gemeinsam durchgeführt werden können, wenn der interne Text des Zielprogramms vorliegt. Diese Idee wurde zum ersten Male in /176/ diskutiert; ähnliche Ideen erschienen ein halbes Jahr später in /192/.

Ein zusätzlicher und sehr wesentlicher Vorteil, den die eben erwähnte gemeinsame Optimierung bietet, besteht darin, daß die bereits vorhandenen Algorithmen zur Reduzierung der Ablaufgraphen genau so gut für die Reduzierung der Programmgraphen angewendet werden können. Dabei ist zu berücksichtigen, daß die Kantenmenge des Programmgraphen sowohl gerichtete als auch ungerichtete Kanten enthält. Unter der Reduktion eines Graphen wird eine systematische Methode verstanden, mit deren Hilfe die einzelnen Knoten des Graphen Schritt für Schritt miteinander zusammengefaßt werden. Nach jedem Schritt erhält man einen neuen Graphen, der aus wenigen Knoten besteht und bei dem die

Gewichte der Knoten entsprechend vergrößert werden. Für ein großes Programm, dessen Programmgraph tausende oder zehntausende von Knoten enthält, ermöglicht es eine vorangehende Reduktion, die Anzahl der Knoten des Speicherorientierten Programmgraphen bis unter eine gegebene obere Grenze (z.B. ein tausend) zu reduzieren. Auf diese Weise läßt sich sowohl die anschließende dynamische Messung (Abs. 4.4.2) als auch die optimale Zerlegung von Netzen mit viel geringerem Aufwand durchführen.

Die Parameter K_M und E_M des Programmgraphen lassen sich im Grunde auch vom Programmierer des entsprechenden Programms angeben. Dabei hat der Programmierer die Möglichkeit, sich auf seine guten Kenntnisse der Ablaufstruktur des eigenen Programms zu stützen. Darüberhinaus kann er selbst bestimmen, wie grob bzw. wie fein die Ablaufstruktur des Programms durch den Programmgraphen dargestellt werden soll. Die Knoten des sich ergebenden Programmgraphen können z.B. ein Unterprogramm, eine Prozedur, ein Teilprogramm oder ein Datenbereich sein. Die Gewichte der Knoten (der Parameter g) lassen sich anschließend mit Hilfe des Compilers bei der Codegenerierungsphase auffinden.

4.4.2 Dynamische Messung

Unter einer dynamischen Messung wird ein Meßverfahren verstanden, bei dem das Zielprogramm in dessen dynamischem Ablauf Ereignisse erzeugt, die von einem Meßprogramm wahrgenommen und registriert werden. Die Messung wird voll-dynamisch genannt, wenn die Ereignisse des laufenden Prozesses, der sich auf das Zielprogramm bezieht, registriert werden. Wird das Zielprogramm jedoch interpretierend ausgeführt, so wird die Messung als halbdynamisch bezeichnet; in diesem Falle wird der laufende Prozeß simuliert. Aus der Folge der Aktivitätsphasen des laufenden Prozesses wird der Parameter f des Programmgraphen abgeleitet. Programme, die in Algol oder PL/I abgefaßt werden, können rekursive Prozeduren enthalten. Bei solchen Prozeduren besteht die Möglichkeit, Speicherplatz dynamisch, d.h. zum Zeitpunkt des Prozeduraufrufs, anzufordern. Für Programme mit dieser Eigenschaft müssen die Knotengewichte des Programmgraphen mittels der dynamischen Messung bestimmt werden.

Der in /175/ beschriebene Meßprozeß ist ein Beispiel zur Durchführung der voll-dynamischen Messung; ein anderes Beispiel ist die "check points"-Methode /135,140,188/. Die halbdynamische Messung ist das älteste, am meisten verwendete Meßverfahren, das zur Messung des Prozeßverhaltens im Adreßraum benutzt wurde /181,189/.

Bei der Durchführung einer dynamischen Messung sieht man sich ständig mit dem Problem der statistischen Absicherung der Meßdaten konfrontiert. Zur Veranschaulichung dieses Problems nehmen wir der Einfachheit wegen an, daß die Kantengewichte des speicherorientierten Programmgraphen (d.h. der Parameter f) gemessen werden sollen, wobei jedes Kantengewicht aus dem statistischen Mittel der gemessenen Übergangshäufigkeiten zwischen den Endknoten der entsprechenden Kante gebildet wird. Das Problem besteht darin, die Messung derart durchzuführen, daß sich ein zuverlässiger statistischer Mittelwert des Parameters f ergibt. Anders ausgedrückt, der gemessene Parameter f soll nicht nur exemplarisch, d.h. allein für die bei der Messung gewählten Datenmengen des Programms, gültig, sondern auch für andere Datenmengen zu verallgemeinern sein. Ohne den statistisch abgesicherten Parameter f wäre das Ergebnis der darauffolgenden optimalen Zerlegung des Programmgraphen nicht für alle möglichen Datenmengen des Programms aussagefähig. Betrachtet man die Durchführung der dynamischen Messung als ein statistisches Experiment, so besteht die Lösung des eben erwähnten Problems darin, das Experiment mit wissenschaftlich fundierter Methode sorgfältig zu planen, derart, daß ein statistisch abgesichertes Ergebnis mit vertretbarem Aufwand erzielt wird. Die wissenschaftlich fundierte Methode für die Planung von Experimenten gehört zu klassischen "Design"-Problemen (z.B. /267/), deren Lösungsverfahren in der mathematischen Theorie der Kombinatorik (z.B. /261,274/) begründet sind.

4.5 Schlußbemerkung

Bei dem in Kapitel 2 entwickelten Angschen Verfahren zur optimalen Zerlegung von Netzen haben wir angenommen, daß die Komponenten der Parameter f und g des Netzes Konstanten sind, obwohl jede von ihnen eigentlich das statistische Mittel einer stochastischen Variablen darstellt. Wenn man diese Einschränkung zum Teil bzw. im ganzen fallen läßt, dann erhält man ein Netz, dessen Kantengewichte und dessen Knotengewichte Verteilungsfunktionen sind. Die optimale Zerlegung eines solchen Netzes stellt eine interessante und herausfordernde Aufgabe dar, die als Fortsetzung der in dieser Arbeit durchgeführten Untersuchungen anzusehen ist. Dabei interessiert man sich insbesondere für die Verteilung der Lösungen, die sich unter den folgenden Voraussetzungen ergeben:

- i) Die Komponenten von f bestehen aus beliebigen bzw. gegebenen Verteilungsfunktionen. Die Komponenten von g sind Konstanten.
- ii) Sowohl die Komponenten von f als auch die Komponenten von g sind beliebige bzw. gegebene Verteilungsfunktionen.

Untersuchungen mit dem Ziel, solche Probleme zu lösen, sind noch kaum in der vorhandenen Literatur zu finden. Bei der Arbeit von Zimmermann und Pollatschek /71/ handelt es sich um die Verteilung des Optimums einer allgemeinen linearen Programmierungsaufgabe mit 0 - 1 Variablen, deren Zielfunktion Koeffizienten enthält, die beliebig verteilt sind. Diese Arbeit läßt sich als Ansatz für die weiteren Untersuchungen betrachten. Es scheint zweckmäßig zu sein, die theoretischen Aspekte des Problems mit den in Abschnitt 4.4 erwähnten Methoden für die Planung von Experimenten (Messungen) gemeinsam zu untersuchen.

5. Literatur

5.1 Verzweigungsprinzip - Theoretische Grundlage und Ideen

- 1 Agin, N Management Science V.13B (1966), 176 -185
"Optimum seeking with branch-and-bound"
- 2 Balas, E Oprn. Res. V.16 (1968), 442 - 445
"A note on the branch-and-bound principle"
- 3 Colomb & Baumert J. ACM V.12 (1965), 516 -524
"Backtrack Programming"
- 4 Geoffrion & Marsten Management Science V.18 (1972), 465 -491
"Integer programming algorithms: a framework and state of the art survey"
- 5 Ibaraki, T Int. J. Comp. and Inf. Sci. V.5 (1976), 315 -344
"Theoretical comparisons of search strategies in branch-and-bound algorithms"
- 6 Ibaraki, T Math. Oprn. Res. V.1 no.3 (1976), 287 - 298
"Computational efficiency of approximate branch-and-bound algorithms"
- 7 Ibaraki, T J. ACM V.24 (1977), 264 -279
"The power of dominance relations in branch-and-bound algorithms"
- 8 Kohler & Steiglitz J. ACM V.21 (1974), 140 -156
"Characterization and theoretical comparison of branch-and-bound algorithms for permutation problems"
- 9 Kohler & Steiglitz Coffman (Ed.): "Computer and job-shop scheduling theory" John-Wiley 1976, Chapter vi, 229 - 287
"Enumerative and iterative computational approaches"
- 10 Lawler & Wood Oprn. Res. V.14 (1966), 699 - 719
"Branch-and-bound methods: A survey"
- 11 Lawler & Bell Oprn. Res. V.14 (1966), 1098 - 1112
"A method for solving discrete optimization problems"
- 12 Mitten, L.G. Oprn. Res. V.18 (1970), 24 - 34
"Branch-and-bound methods: General formulation and properties"
- 13 Noltenmeier, H Computing V.8 (1971), 99 -106
"Ein branch-and-bound-Verfahren-Generator"

- 14 Rinnooy Kan A.H.G. Oprn. Res. V.24 (1976), 1176 -1178
"On Mitten's axioms for branch-and-bound"
- 15 Salikin, H.M. Oprn. Res. V.18 (1970), 549 -554
"On the merit of the generalized origin and restarts in implicit enumeration"
- 16 Schoch, M VEB Deutscher Verlag d. Wissenschaften, Berlin, 1976
"Das Erweiterungsprinzip"
- 17 Tang & Wong Ann. Princ. Conf. Inf. Syst. Sci. 1972, 641 -644
"Improving the branch-and-bound method"
- 18 Walker, R.S. Amer. Math. Soc. Symp. Appl. Math. V.10, 91 - 94
"An enumerative technique for a class of combinatorial problems"
- 19 Weinberg, F. (Hrsg.) Springer Verlag 1968
"Einführung in die Methode branch-and-bound"

5.2 Verzweigungsprinzip - Anwendungen und relevante Verfahren

- 20 Ahrens & Finke Oprn. Res. V.23 (1975), 1099 -1109
"Merging and sorting applied to the zero-one knapsack problem"
- 21 Bonzon, P. J. ACM V.17 (1970), 675 - 682
"Necessary and sufficient conditions for dynamic programming of combinatorial type"
- 22 Cabot, A.V. Oprn. Res. V.18 (1970), 306 - 311
An enumeration algorithm for knapsack problems"
- 23 Dakin, R.J. Computer J. V.8 (1965), 250 -255
"A tree search algorithm for mixed integer programming problems"
- 24 Duncan & Scott Oprn. Res. V.23 (1975), 240 - 259
"A branch-and-bound algorithm for pagination"
- 25 Fayard & Plateau Math. Programming V.8 (1975), 272 - 307
"Resolution of the 0 - 1 Knapsack Problem: Comparison of methods"
- 26 Fillmore & Williamson SIAM J. Computing V.3 (1974), 41 - 55
"On backtracking: A combinatorial description of the algorithm"

- 27 Garfinkel & Nemhauser Management Science V.16 (1970), B495 - B508
"Optimal political districting by implicit enumeration techniques"
- 28 Geoffrion, A.M. SIAM Rev. V.9 (1967), 178 - 190
"Integer programming by implicit enumeration and Balas method"
- 29 Geoffrion, A.M. Oprn. Res. V.17 (1969), 437 - 454
"An improved implicit enumeration approach for integer programming"
- 30 Gilmore, P.C. SIAM J. Appl. Math. V.10 (1962), 305 - 313
"Optimal and suboptimal algorithms for quadratic assignment problems"
- 31 Greenberg & Hegerich Management Science V.16 (1970), 327 - 332
"A branch search algorithm for the knapsack problems"
- 32 Help & Karp SIAM J. Appl. Math. V.10 (1962), 196 - 210
"A dynamic programming approach to sequencing problems"
- 33 Ignall & Schrage Oprn. Res. V.13 (1965), 410 - 412
"Applications of the branch-and-bound technique to some flowshop scheduling problems"
- 34 Ingargiola & Korsh Oprn. Res. V.23 (1975), 1110 - 1119
"An algorithm for the solution of 0 - 1 loading problem"
- 35 Kolesar, P.J. Management Science V.13 (1967), 723 - 735
"A branch-and-bound algorithm for the knapsack problem"
- 36 Land & Doig Econometrica V.28 (1960), 497 - 520
"An enumerative method for solving discrete programming problems"
- 37 Lawler, E.L. Management Science V.9 (1963), 586 - 599
"The quadratic assignment problem"
- 38 Little et al. Oprn. Res. V. 11 (1963), 972 - 989
"An algorithm for the traveling salesman problem"
- 39 Michaud, P. IBM J. Res. Dev. Nov. 1972, 573 - 578
"Exact implicit enumeration method for solving the set partitioning problem"
- 40 Ross & Soland Math. Programming V.8 (1975), 91 - 103
"A branch-and-bound algorithm for the generalized assignment problem"

- 41 Schrage, L. Oprn. Res. V.18 (1970), 263 - 278
"Solving resource-constrained network problems by implicit enumeration - non-preemptive case"
- 42 Schrage, L. Oprn. Res. V.20 (1972), 668 - 677
"Solving resource-constrained network problems by implicit enumeration - preemptive case"
- 5.3 Kombinatorische Programmierung mit 0 - 1 Variablen
- 43 Balas, E. Oprn. Res. V.13 (1965), 517 - 549
"An additive algorithm for solving linear programs with 0 - 1 variables"
- 44 Balas, E. Oprn. Res. V.15 (1967), 915 - 957
"Discrete programming by the filter method"
- 45 Balas & Padberg Oprn. Res. V.20 (1972), 1152 - 1161
"On the set-covering problem"
- 46 Balas & Padberg Oprn. Res. V.23 (1975), 74 - 90
"On the set-covering problem II: An algorithm for set-partitioning"
- 47 Balas & Padberg SIAM Rev. V.18 (1976), 710 - 760
"Set partitioning: a survey"
- 48 Burge, W.H. IBM J. Res. Dev. Sept. 1972, 450 - 461
"Combinatorial programming and combinatorial analysis"
- 49 Ellwein, L.B. Oprn. Res. V.22 (1974), 144 - 150
"A flexible enumeration scheme for zero-one programming"
- 50 Fleischmann, B. Oprn. Res. V.15 (1967), 153 - 155
"Computational experience with the algorithm of Balas"
- 51 Freeman, R.J. Oprn. Res. V.14 (1966), 935 - 941
"Computational experience with a "Balasian" integer programming algorithm"
- 52 Garfinkel & Nemhauser Oprn. Res. V.17 (1969), 848 - 856
"The set-partitioning problem: set covering with equality constraints"
- 53 Geoffrion, A.M. (Ed.) Addison-Wesley, Reading, Mass., 1972
"Perspectives on optimization: a collection of expository articles"

- 54 Glover & Zionts Oprn. Res. V.13 (1965), 546 - 549
"A note on the additive algorithm of Balas"
- 55 Glover, F. Oprn. Res. V.13 (1965), 879 - 919
"A multiphase-dual algorithm for the zero-one integer programming problems"
- 56 Glover, F. Oprn. Res. V.16 (1968), 741 - 749
"Surrogate constraints"
- 57 Gue, Liggeit & Cain C. ACM V.11 (1968), 837 - 844
"Analysis of algorithms for the zero-one programming problem"
- 58 Hammer & Rudeanu Springer-Verlag, Berlin-New York 1968
"Boolean methods in operations research and related areas"
- 59 Hammer & Peled J. ACM V.19 (1972), 265 - 282
"On the maximization of a pseudo-boolean function"
- 60 Hammer & Rosenberg Zaremba (Ed.) Acad. Press, N.Y. 1972, 453-463
"Appl. of numb. theory to numerical analysis"
"Equivalent form of 0 - 1 Programs"
- 61 Hammer P.L. Hammer & Zoutendijk (Ed.), North Holland, 1974
"Math. Progr. in theory and practice", 311-363
"Boolean procedures for bivalent programming"
- 62 Hammer, P.L. Roy (Ed.), Proc. Nato Adv.St. Inst., Versailles, 74
"Combinatorial Programming: methods & applications"
"Boolean elements in combinatorial optimization"
- 63 Hillier & Connors Management Science V.13 (1966), 42 - 57
"Quadratic assignment problem algorithms and the location of indivisible facilities"
- 64 Kral. J. Kybernetika V.4 (1968), 6 - 11
"The formulation of the problem of program segmentation in the terms of pseudoboollean programming"
- 65 Lawler, E. L. SIAM J. Appl. Math. V.14 (1966), 1115 - 1132
"Covering problems: duality relations and a new method of solution"
- 66 Lawler, E.L. Management Science V.9 (1963), 586 -599
"The quadratic assignment problem"

- 67 Lemke & Spielberg Oprn. Res. V.15 (1967), 892 - 914
"Direct search algorithms for zero-one and mixed integer programming"
- 68 Marsten, R.E. Management Science V.20 (1974), 774 - 787
"An algorithm for large set partitioning problems"
- 69 Pierce, J.F. Management Science V.15 (1968), 191 - 209
"Applications of combinatorial programming to a class of all zero-one integer programming problems"
- 70 Pierce & Lasky Management Science V.19 (1973), 528 - 543
"Improved combinatorial programming algorithms for a class of all-zero-one integer programming problems"
- 71 Zimmermann & Pollatschek Oprn. Res. V.23 (1975), 137 - 149
"The probability distribution function of the optimum of a 0 - 1 linear program with randomly distributed coefficients of the objective functions and the right hand side"

5.4 Heuristische Verfahren zur Lösung kombinatorischer Optimierungsaufgaben

- 72 Celebiler, M.I. Oprn. Res. V. 17 (1969), 993 - 1004
"A probabilistic approach to solving assignment problems"
- 73 Chandra et al. Theor. Comp. Sci. TSCDI 3(3) 1976, 293 - 304
"Approximate algorithms for some generalized knapsack problems"
- 74 Eilon & Christofides Management Science V.17 (1971), 259 - 268
"The loading problem"
- 75 Graves & Whinston Management Science V.17 (1970), 453 - 471
"An algorithm for the quadratic assignment problem"
- 76 Horowitz & Sahni J. ACM V.21 (1974), 277 - 292
"Computing partitions with applications to the Knapsack problem"
- 77 Ibarra & Kim J. ACM V.22 (1975), 463 - 468
"Fast approximate algorithms for the knapsack and sum of subset problems"
- 78 Ingargiola & Korsh Oprn. Res. V.23 (1975), 1110 - 1119
"An algorithm for the solution of the 0 - 1 loading problem"
- 79 Johnson, D.S. MIT MAC-TR-109 June 1973
"Near-optimal BIN packing algorithms"

- 80 Johnson, D.S. J. Comp. Syst. Sci. V.8 (1974), 272 - 314
"Fast algorithms for BIN-Packing"
- 81 Johnson, D.S. J. Comp. Syst. Sci. V.9 (1974), 256 - 278
"Approximate algorithms for the combinatorial problems"
- 82 Krone & Steiglitz Oprn. Res. V.22 (1974), 629 - 638
"Heuristic programming solution of a flowshop scheduling problem"
- 83 Lenstra, J.K. Oprn. Res. V.22 (1974), 413 - 414
"Clustering a data array and the traveling-salesman problem"
- 84 Lin, S. Bell Syst. Tech. J. V.44 (1965), 2245 - 2269
"Computer solutions of the traveling salesman problem"
- 85 Lin & Kernighan Ann. Princ. Conf. Inf. Syst. Sci. 1971, 210 - 213
"A heuristic technique for solving a class of combinatorial optimization problems"
- 86 Lin & Kernighan Oprn. Res. V.21 (1973), 498 - 516
"An effective heuristic algorithm for the traveling salesman problem"
- 87 McCormick et al. Oprn. Res. V.20 (1972), 993 - 1009
"Problem decomposition and data organisation by a clustering technique"
- 88 Reiter & Sherman SIAM J. Appl. Math. V.13 (1965), 864 - 889
"Discrete optimization"
- 89 Roth, R.H. J. ACM V.17 (1970), 303 - 313
"An approach to solving linear discrete optimization problems"
- 90 Sahni, S. J. ACM V.22 (1975), 115 - 124
"Approximate algorithms for the 0/1 knapsack problem"
- 91 Sahni, S. Oprn. Res. V.25 (1977), 920 - 936
"General techniques for the combinatorial approximation"
- 92 Savage, S.L. Math. Programming V.10 (1976), 345 - 366
"Some theoretical implications of local optimization"

5.5 Komplexität der kombinatorischen Optimierungsaufgaben und die Analyse von Algorithmen

- 93 Aho, Hopcroft & Ullman Addison-Wesley, Reading, Mass, 1974
"The design and analysis of computer algorithms"
- 94 Cook, S.A. Proc. 3rd ACM Symp. Theory of comp. 1971, 151 - 158
"The complexity of theorem proving procedures"
- 95 Edmonds & Karp J. ACM V.19 (1972), 248 - 264
"Theoretical improvements in the algorithmic efficiency of network problems"
- 96 Frazer, W.D. Proc. AFIPS V.40 (1972) SJCC, 483 - 491
"Analysis of combinatorial algorithms - a sample of current methodology"
- 97 Garey et al. Proc. 4th ACM Symp. Theory of Comp. 1972, 143 - 150
"Worst case analysis of memory allocation algorithms"
- 98 Garey & Johnson Theoretical comp. Sci.1, 3 (1976), 237 - 267
"Some simplified NP-complete graph problems"
- 99 Garey & Johnson Traub (Ed.), Academic press, N.Y., 1976, 41 - 52
"Algorithms and complexity: new direc. & results"
"Approximation algorithms for combinatorial problems: an annot. bibliogr."
- 100 Garey et al. Math. Oprn. Res. V. 1 (1976), 117 - 129
"The complexity of flowshop & jobshop scheduling"
- 101 Graham, R.L. Coffman (Ed.), John wiley, N.Y., 1976, Chap. 5
"Computer and job-shop scheduling theory"
"Bounds on the performance of scheduling algorithms"
- 102 Hyafil & Rivest IRIA Rapport de Recherche no. 33, Oct. 1973
"Graph partitioning and constructing optimal decision trees are polynomial complete problems"
- 103 Johnson et al. SIAM J. Computing V.3 (1974), 299 - 326
"Worst case performance bounds for simple one-dimensional packing algor."
- 104 Karp, R.M. Muller & Thatcher (Ed.), Plenum Press, 1972, 85-104
"Complexity of computer computations"
"Reducibility among combinatorial problems"

- 105 Knuth, D.E. Proc. IFIP Congress 1971, Vol 1, North-Holland Co.
135 - 143
"Mathematical analysis of algorithms"
- 106 Papadimitriou et al. SIAM J. Computing 1977, 76 - 83
"On the complexity of local search for the traveling salesman problem"
- 107 Reingold, E.M. Proc. AFIPS V.40 (1972) SJCC, 471 - 481
"Establishing lower bounds on algorithms: a survey"
- 108 Sahni, S. SIAM J. Computing V. 3 (1974), 262 - 279
"Computationally related problems"
- 109 Sahni & Gonzaley J. ACM V.23 (1976), 555 - 565
"P-complete approximation problems"
- 110 Ullman, J.D. J. Comp. & Syst. Sci. V.10 (1975), 384 - 393
"NP-complete scheduling problems"
- 111 Weide, B. Comp. Survey V.9 (1977), 291 - 313
"A survey of analysis techniques for discrete algorithms"

5.6 Optimale Zerlegung von Netzen

- 112 Ang, H.C. (H. Widjaja) Interne Arbeitsnotiz SS-77, 14 Juli 1977
"Optimale Zerlegung von Graphen und die relevanten kombinatorischen Optimierungsaufgaben"
- 113 Baer & Caughey Proc. AFIPS V.40 (1972) SJCC, 23 - 36
"Segmentation and optimization of programs from cyclic structure analysis"
- 114 Christofides & Brooker SIAM J. Appl. Math. V.30 (1976), 55 - 69
"The optimal partitioning of graphs"
- 115 Gorinshteyn, L.L. Engineering Cybernetics V.1 (1969), 76 - 82
"The partitioning of graphs"
- 116 Jensen, P.A. Oprn. Res. V.19 (1971), 916 - 932
"Optimal network partitioning"
- 117 Jensen, R.E. Oprn. Res. V.17 (1969), 1034 - 1057
"A dynamic programming algorithm for cluster analysis"
- 118 Kernighan, B.W. J. ACM V.18 (1971), 34 - 40
"Optimal sequential partitions of graphs"

- 119 Kernighan & Lin Bell Syst. Tech. J. V.49 (1970), 291 - 307
"An efficient heuristic procedure for partitioning graphs"
- 120 Kundu & Misra SIAM J. Computing V.6 (1977), 151 - 154
"A linear tree partitioning algorithm"
- 121 Lawler, E.L. IRE Trans. EC - 11 (1962), 86 - 88
"Electric assemblies with a minimum number of interconnections"
- 122 Luccio & Sami IEEE Trans. CT - 16 (1969), 184 - 188
"On the decomposition of networks in minimally interconnected subnetworks"
- 123 Lukes, J.A. IBM J. Res. Dev. V. 18 (1974), 217 - 224
"Efficient algorithm for the partitioning of trees"
- 124 Lukes, J.A. IBM J. Res. Dev. V. 19 (1975), 170 - 180
"Combinatorial solution to the partitioning of general graphs"
- 125 Paige, M.R. IEEE Trans. SE - 3 (1977), 386 - 393
"On partitioning program graphs"
- 126 Ramamoorthy, C.V. J. ACM V.13 (1966), 211 - 222
"Analysis of graphs by connectivity considerations"
- 127 Schindler & Wagner 3rd GI Jahrestagung, Hamburg, Okt. 1973
"Optimale Zerlegung von Computer Programmen durch Zerlegung von Graphen"
- 128 Spies, P.P. Univ. Bonn, Institut für Informatik, Abt. Inf. II
Briefwechsel: 4.10.77; 10.10.77; 18.10.77

5.7 Automatische Programm-Analyse

- 129 Baer & Estrin Proc. Hawaii Int. Conf. Syst. Sci. 1969, 139 - 142
"Frequency numbers associated with directed graph representations of computer programs"
- 130 Bernstein IEEE Trans. EC - 15 (1966), 757 - 763
"Analysis of programs for parallel computation"
- 131 Bredt, T.H. TR no. 6 STAN-CS-70-170 May 1970, Stanford Univ.
"Analysis and Synthesis of Concurrent Sequential Programs"

- 132 Gosden, J.A. Proc. AFIPS V.29 FJCC (1966), 650 - 660
"Explicit parallel processing description and control in programs for multi- and uni-processor computers"
- 133 Hennel et al. Inf. Proc. Letters V.5 (1976), 136 - 140
"On program analysis"
- 134 Ingalls, D. Rustin (Ed.), Prentice Hall Inc. (1972), 107 - 128
"Design and optimization of compilers"
"The execution time profile as a programming tool"
- 135 Knuth, D.E. Software Prac. & Expr. V.1 (1971), 105 - 133
"An empirical study of Fortran-Programs"
- 136 Knuth & Stevenson BIT 13 (1973), 313 - 322
"Optimal measurement points for program frequency counts"
- 137 Krider, L. J. ACM V.11 (1964), 429 - 436
"A flow analysis algorithms"
- 138 Ramamoorthy & Ho IEEE Trans. SE - 1 (1975), 46 - 58
"Testing large software with automated software evaluation systems"
- 139 Robinson & Torsun Computer J. V.19 (1976), 56 - 62
"An empirical analysis of Fortran programs"
- 140 Russel, E.C. UCLA-Rept 69-12 March 1969, Univ. Calif. Los Ang.
"Program analysis by digital computer"
- 141 Wegbreit, B. C. ACM V.18 (1975), 528 - 539
"Mechanical program analysis"
- 142 Wilde, D.U. MIT MAC-TR-43 1967
"Program analysis by digital computer"

5.8 Programm-Optimierung und Ablaufgraphen

- 143 Adams et al. SIAM J. Computing V.3 (1974), 222 - 223
"A note on the Hecht Ullman characterization of non-reducible flow graphs"
- 144 Aho & Ullman Prentice Hall Inc. Englewood Cliffs N.J. 1973
"The theory of parsing, translation and compiling Vol. II: Compiling"
- 145 Allen, F.E. Ann. Rev. in aut. progr. V.5 (1969), 239 - 307
"Program optimization"

- 146 Allen, F.E. Proc. IFIP 1974, 398 - 402
"Interprocedural data flow analysis"
- 147 Allen & Cocke Rustin (Ed.): "Design and optimization of
compilers", Prentice Hall Inc. 1972
"A catalogue of optimizing transformations"
- 148 Cocke, J. ACM SIGPLAN Notices V.5 (1970), 20 - 24
"Global common subexpression elimination"
- 149 Earnest et al. J. ACM V.19 (1972), 23 - 42
"Analysis of graphs by ordering of nodes"
- 150 Earnest, C. J. ACM V.21 (1974), 76 - 102
"Some topics in code optimization"
- 151 Graham & Wegman J. ACM V.23 (1976), 172 - 202
"A fast and usually linear algorithm for global flow analysis"
- 152 Hecht & Ullman SIAM J. Computing V.1 (1972), 188 - 202
"Flow graph reducibility"
- 153 Hecht & Ullman J. ACM V.21 (1974), 367 - 375
"Characterization of reducible flow graphs"
- 154 Ianov, Y.I. C. ACM V.1 (1958), 8 - 12
"On the equivalence and transformation of program scheme"
- 155 Kennedy, R. Int. J. Comp. Math. V.3 (1971), 5 - 15
"A global flow analysis algorithm"
- 156 Lowry & Medlock C. ACM V.12 (1969), 13 - 22
"Object code optimization"
- 157 Nunamaker et al. Tou (Ed.): "Information systems" COINS IV
Plenum Press N.Y. 1974
"Processing system optimization through automatic design and
reorganisation of program modules"
- 158 Rutledge, J.D. J. ACM V.12 (1964), 1 - 9
"On Ianov's program schemata"
- 159 Schaeffer, M. Prentice Hall Inc. Englewood Cliffs N.J. 1973
"A mathematical theory of global program optimization"

- 160 Tarjan, R.E. J. Comp. Syst. Sci. V.9 (1974), 355 - 365
 "Testing flow graph reducibility"

5.9 Strukturierung von Programmen und Programm-Verifikation

- 161 Ashcroft & Mana SIAM J. Computing V.4 (1975), 125 - 146
 "Translating program schemes to while-schemes"
- 162 Böhm & Jacopini C. ACM V.9 (1966), 366 - 371
 "Flow diagrams, Turing machines and languages with only two formation rules"
- 163 Dijkstra, E.W. C. ACM V.11 (1968), 147 - 148
 "Letter to the editor: "Goto statement considered harmful""
- 164 Fisher, D.R. AFOSR-70-1846-TR Carnegie-Mellon Univ. May 1970
 "Control structures for programming languages"
- 165 Floyd, R.W. Amer. Math. Soc. Symp. Appl. Math. Proc. V.19, 1967
 "Assigning meanings to programs"
- 166 Hoare, C.A.R. C. ACM V.12 (1969), 576 - 583
 "An axiomatic basis for computer programming"
- 167 Kasai, T. J. Comp. Syst. Sci. V.9 (1974), 177 - 195
 "Translatability of flowcharts into while programs"
- 168 Kosaraju, S.R. J. Comp. Syst. Sci. V.9 (1974), 232 - 255
 "Analysis of structured programs"
- 169 Ledgard & Marcotty C. ACM V.18 (1975), 629 - 638
 "A genealogy of control structures"
- 170 Lipton et al. J. ACM V.23 (1976), 720 - 732
 "Space and time hierarchies for classes of control structures and
 Data structures"
- 171 Luckham et al. J. Comp. Syst. Sci. V.4 (1970), 220 - 249
 "On formalised computer programs"
- 172 Peterson et al. C. ACM V.16 (1973), 503 - 512
 "On the capabilities of while, repeat and exit statement"
- 173 Urschler, G. IBM J. Res. Dev. V.19 (1975), 181 - 194
 "Automatic structuring of programs"

5.10 Programmverhalten, Speicher- und Dateiverwaltung

- 174 Ang, H.C. (H. Widjaja) Interner Bericht des IMMD III, 31.7.75
"Bericht zur Speicherverwaltung"
- 175 Ang, Klar, Schreiber Arbeitsbericht des IMMD, Band 8 No. 9, Dez. 75
"Messungen mit dem Zählmonitor 2"
- 176 Ang, H.C. (H. Widjaja) Interner Bericht des IMMD III, 9.2.76
"Lokalitätsorientierte Optimierung von Programm- und Datenzugriffen
im Adreßraum"
- 177 Ang, H.C. (H. Widjaja) Interne Arbeitsnotiz IMMD III, SS/1976, 10.8.76
"Programmverhalten in Rechensystemen, Warteschlangennetze, Speicher-
verwaltung - Erkenntnisse, Ideen und Überlegungen"
- 178 Ang, H.C. (H. Widjaja) Interne Arbeitsnotiz IMMD III, WS-76/77, 2.3.77
"Dynamisches Programmverhalten in Modellen der Rechensysteme"
- 179 Babad, J.M. C. ACM V.20 (1977), 22 - 31
"A record and file partitioning model"
- 180 Batson & Brundage Proc. SIGMETRICS 74 Conf., Montreal, 121 - 126
"Measurements of the virtual memory demands of Algol-60 programs"
- 181 Belady, L.A. IBM Syst. J. V.5 (1966), 78 - 101
"A study of replacement algorithms for a virtual storage computer"
- 182 Buchholz, W. IBM Syst. J. V.2 (1963), 86 - 111
"File organisation and adressing"
- 183 Burkhardt & Chow PIT-CS-BU-72-03 Univ. Pittsburg, Dec. 1972
"A survey of replacement algorithms in virtual memory operating systems"
- 184 Courtois & Vantilborgh Acta Informatica V.6, F3 (1976), 251 - 275
"A decomposable model of program paging behavior"
- 185 Denning, P.J. Comp. Surveys V.2 (1970), 153 - 190
"Virtual memory"
- 186 Dittmar, M. Dipl. Arbeit des IMMD III, Dez. 1976
"Untersuchung einer zweistufigen Speicherhierarchie unter Annahme
assoziativer Logik im Primärspeicher"

- 187 Eisner & Severance J. ACM V.23 (1976), 619 - 635
"Mathematical techniques for efficient record segmentation in large shared data bases"
- 188 Ferrari, D. IBM J. Res. Dev. V.19 (1975), 244 - 257
"Tailoring programs to models of program behavior"
- 189 Fine et al. 21st ACM Nat. Conf. (1966), 223 - 228
"Dynamic program behavior under paging"
- 190 Freiburger et al. IBM J. Res. Dev. V.19 (1975), 230 - 243
"Patterns in program references"
- 191 Greenberg, B.S. MIT MAC-TR-127 May 1974
"An experimental analysis of program reference patterns in the Multics virtual memory"
- 192 Innes & Tsur Inf. Proc. Letters V.5 (1976), 91 - 96
"Interval analysis, pagination and program locality"
- 193 Johnson, J.W. MIT MAC-TR-148 March 1975
"Program restructuring for virtual memory systems"
- 194 Karp, McKellar, Wong SIAM J. Computing V.4 (1975), 271 - 286
"Near-optimal solutions to a two dimensional placement problem"
- 195 Kuck & Lawrie Report no. 363 Dec. 1969 Illinois Univ.
"The use and performance of memory hierarchies: a survey"
- 196 Lange, M. Lect. Notes Comp. Sci. V.13, Springer Verl. 133-137
"Software-Messungen von Programm-Parametern unter Master"
- 197 Lowe, T.C. TR-73-1558-1 Informatics Inc. Aug. 1973
"Final report on investigations of boundary crossing in partitioned information systems"
- 198 Madnick, S.E. MIT MAC-TR-107 April 1973
"Storage hierarchy systems"
- 199 Maurer & Lewis Comp. Surveys V.7 (1975), 5 - 19
"Hash Table Methods"
- 200 Nievergelt, J. Comp. Surveys V.6 (1974), 195 - 207
"Binary search trees and file organisation"

- 201 Opderbeck, H. UCLA-ENG-7418 UCLA April 1974
"measurement and modeling of program behavior and its applications"
- 202 Peters, C.B. RADC-TR-71-231 Informatics Inc. Nov. 1971
"Experiments in automatic paging Vol. I & Vol. II"
- 203 Pinkerton, T.B. Tech. Rept. no. 4 Univ. of Michigan, April 1968
"Program behavior and control in virtual computer systems"
- 204 Randell & Kuehner C. ACM V.11 (1968), 297 - 306
"Dynamic storage allocation systems"
- 205 Rau, B.R. TR no. 95, Standford electr. Lab. July 1975
"The stack working set: a characterization of spatial locality"
- 206 Ray-chaudhuri, D.K. SIAM J. Appl. Math. V.16 (1968), 973 - 992
"Combinatorial information retrieval systems for files"
- 207 Rosenberg, A.L. J. Comp. Syst. Sci. V.5 (1971), 193 - 238
"Data Graphs and addressing schemes"
- 208 Rosenberg, A.L. IBM RC-5798 (no. 25127), 1.9.76
"Storage mappings for extensible arrays"
- 209 Sampson, P.D. Rept. no. 28. Pattern analysis, Brown Univ. July 72
"Regime behavior in page referencing patterns of computer programs"
- 210 Schwetman, H.W. TSN-12 Univ. of Texas at Austin, July 1970
"A study of resource utilization and performance evaluation of large
scale computer systems"
- 211 Sekino, A MIT MAC-TR-103 September 1972
"Performance evaluation of multiprogrammed time-shared computer systems"
- 212 Senko et al. IBM research Lab. RADC-TR-70-90 June 1970
"Formatted file organisation techniques"
- 213 Severance, D.G. Comp. Surveys V.6 (1974), 175 - 194
"Identifier search mechanisms: a survey and generalized model"
- 214 Smith, A.J. SIAM J. Computing V.5 (1976), 743 - 757
"Analysis of the optimal, look-ahead demand paging"

- 215 Svobodova, L. TR no. 72 SEL 74-036 Stanford Univ. June 1974
"Computer performance measurement and evaluation methods: analysis and applications"
- 216 Trivedi, K.S. UIUCDCS-R-74-662 Univ. of Illinois, July 1974
"Prepaging and applications to structured array problems"
- 217 Woolf, A.M. RADC-TR-71-165 Univ. of Michigan, Aug. 1971
"Analysis and optimization of multiprogrammed computer system using storage hierarchies"
- 218 Yelowitz, L. Proc. Symp. on computer & Automata V.XXI, 225 - 236
Polytechnic Inst. of Brooklyn, April 1971
"Toward optimal pagination for structured programs"

5.11 Rechnerarchitektur, Betriebssysteme, Prozeßmodelle

- 219 Baer, J.L. Rept. no. 68-46 UCLA, Oct. 1968
Graph models of computation in computer systems"
- 220 Belpaire & Wilmotte Goos & Hartmanis (Ed.): "Operating systems"
Springer Verlag, 1974, Rocquincourt
"Correctness of realizations of levels of abstraction in operating systems"
- 221 Bergin, G.P. Proc. AFIPS 1964 FJCC Part I, 45 - 55
"Method of control for re-entrant programs"
- 222 Bremer, J.W. IEEE Computer V.9 (1970), 31 - 36
"Hardware Technology in the year 2001"
- 223 Chang, H.C. AFOSR-TR-75-1674 Carnegie-Mellon Univ. June 1975
"Analysis of deadlock avoidance schemes and resource utilization for non-preemptible resources"
- 224 Dijkstra, E.W. C. ACM V.11 (1968), 341 - 346
"The structure of the "THE"-Multiprogramming system"
- 225 Dijkstra, E.W. Genuys (Ed.): "Programming Languages", 43 - 112
Academic Press, 1968
"Cooperating sequential processes"
- 226 Dijkstra, E.W. Hoare (Ed.): "Operating System Techniques"
Academic Press, London, 1972, 72 - 93
"Hierarchical ordering of sequential processes"

- 227 Enslow, P.H. IEEE Computer V.11 (1978), 13 - 21
"What is a "distributed" data processing system"
- 228 Fuller et al. Proc. IEEE V.66 (1978), 216 - 228
"Multi-Microprocessors: an overview and working examples"
- 229 Genrich & Lautenbach Acta Informatica V.2 (1973), 143 - 161
"Synchronizationsgraphen"
- 230 Gilbert & Chandler C. ACM V.15 (1972), 427 - 437
"Interference between communicating parallel processes"
- 231 Händler et al. Workshop of the GI: "Computer Architecture"
Erlangen, May 1975
"A general purpose array with a broad spectrum of applications"
- 232 Händler, W. Arbeitsbericht des IMMD, Band 9, No. 8, 1976, 1 - 3
"Zur Genealogie, Struktur und Klassifizierung von Rechnern"
- 233 Händler, W. Feilmeier (Ed.): "Parallel Computers - Parallel Math." Int. Assoc. for Math. and Comp. in Simul. 77
"Aspects of parallelism in computer architecture"
- 234 Händler, W. Skript einer Vorlesung, IMMD III
"Rechnerarchitektur I und II"
- 235 Hansen, P.B. Prentice Hall Inc. Englewood Cliffs, 1973
"Operating System Principles"
- 236 Hoare, C.A.R. C. ACM V.17 (1974), 549 - 557
"Monitors: an operating system structuring concept"
- 237 Hofmann, F. Arbeitsbericht des IMMD, Band 9, No. 8, 1976, 233-246
"Prozeßmodelle aus der Sicht des Betriebssystementwurfs"
- 238 Holt & Commoner Proj. MAC Conf. on concurrent systems and parallel computations, ACM Conf. Record, 1970, 3 - 52
"Events and conditions"
- 239 Horning & Randell Comp. Surveys V.5 (1973), 5 - 30
"Process structuring"
- 240 Hotz, G. Verlag B.G. Teubner, Stuttgart, 1972
"Informatik: Rechenanlagen Struktur und Entwurf"

- 241 Johnston, J.B. ACM SIGPLAN Notices V.6 (1971), 55 - 82
"The contour model of block-structured processes"
- 242 Karp & Muller J. Comp. Syst. Sci. V.3 (1969), 147 - 195
"Parallel program schemata: a math. model for parallel computation"
- 243 Keller, R.M. Comp. Surveys V.7 (1975), 177 - 195
"Look-Ahead Processors"
- 244 Kosaraju, S.R. ACM Op. Syst. Rev. V.7 (1973), 122 - 126
"Limitations of Dijkstra's semaphore primitive and petri nets"
- 245 Lipton et al. Automata and switching theory, 1974, 145 - 155
"A comparative study of models of parallel computation"
- 246 Martin, D.F. Rept. no. 66-4 Jan. 1966, UCLA
"The automatic assignment and sequencing of computations on parallel processing systems"
- 247 Patil, S.S. MIT MAC-TR-72 May 1970
"Coordination of asynchronous events"
- 248 Peterson & Brecht Proc. IFIP 1974, North Holland Co. 466 - 470
"A comparison of models of parallel computation"
- 249 Randell & Zurcher Proc. IFIP 1968, V.II, North Holland Co. 867 - 871
"Iterative multilevel modelling: a methodology for comp. syst. design"
- 250 Reiter, R RADC-TR-68-350 Univ. of Michigan, Nov. 1968
"A study of model for parallel computations"
- 251 Riddle, W.E. Proc. ACM 1972 Nat. Conf., 1105 - 1117
"Hierarchical modeling of operating systems structure and behavior"
- 252 Rodriguez, J.E. MIT MAC-TR-56 September 1969
"A graph model for parallel computations"
- 253 Rumbaugh, J. IEEE Trans. C-26 (1977), 138 - 146
"A data flow multiprocessor"
- 254 Saltzer, J.H. MIT MAC-TR-30 July 1966
"Traffic control in multiplexed computer system"
- 255 Schlembach, H. Arbeitsnotiz no. 16 IMMD III, 10.6.71
"Deadlock - Probleme"

- 256 Schmid, H.A. Acta Informatica V.16 (1976), 227 - 249
 "On the efficient implementation of conditional critical regions
 and the construction of monitors"
- 257 Shaw & Weiderman Software Prac. & Exper. V.5 (1975), 245 - 267
 "A multiprogramming nucleus with dynamic resource facilities"
- 258 Slutz, D.R. MIT MAC-TR-53 September 1968
 "The flow graph schemata model of parallel computation"
- 259 Vanderbilt, D.H. MIT MAC-TR-67 Oct. 1969
 "Controlled information sharing in a computer utility"
- 260 Wirth, N. C. ACM V.12 (1969), 489 - 498
 "On multiprogramming, machine coding and computer organisation"
- 5.12 Allgemeine Hilfsmittel
- 261 Anderson, I. Clarendon Press, Oxford, 1974
 "A first course in combinatorial mathematics"
- 262 Berge, C. John-Wiley & Sons Inc. 1966
 "The theory of graphs and its applications"
- 263 Berztiss, A.T. Academic Press, N.Y. 1971
 "Data structures: theory and practice"
- 264 Bobrow & Arbib W. B. Saunders Comp., London, 1974
 "Discrete Mathematics: Applied algebra for computer and information science"
- 265 Busacker & Saaty Oldenbourg Verlag, München-Wien, 1968
 "Endliche Graphen und Netzwerke"
- 266 Buzen, J.P. ESD-TR-71-345 Harvard Univ. August 1971
 "Queueing network models of multiprogramming"
- 267 Cochran & Cox Charles E. Tuttle Co., Tokyo, Mod. Asia Ed. 1959
 "Experimental Designs"
- 268 Coffman, E.G. (Ed.) John-Wiley & Sons, N.Y. 1976
 "Computer and job-shop scheduling theory"
- 269 Conway et al. Addison Wesley, Reading, Mass. 1967
 "Theory of scheduling"

- 270 Courtois, P.J. Academic Press, N.Y. 1976
"Decomposability: Queueing and computer system application"
- 271 Crane & Iglehart J. ACM V.21 (1974), 103 - 113
"Simulating stable stochastic systems I: General multiserver queues"
- 272 Frank & Frisch Addison-Wesley, Reading, Mass., 1971
"Communication, Transmission and Transportation Networks"
- 273 Kleinrock, L. John Wiley & Sons 1975
"Queueing Systems Vol. 1: Theory"
- 274 Liu, C.L. McGraw-Hill N.Y. 1968
"Introduction to combinatorial mathematics"
- 275 Murty, K.G. John-Wiley & Sons N.Y. 1976
"Linear and Combinatorial Programming"
- 276 Nilsson, N.J. McGraw-Hill Co. N.Y. 1971
"Problem-solving methods in artificial intelligence"
- 277 Noltenmeier, H. Walter de Gruyter, Berlin, 1976
"Graphentheorie mit Algorithmen und Anwendungen"
- 278 Seshu & Reed Addison-Wesley, Reading, Mass. 1961
"Linear Graphs and Electrical Networks"
- 279 Slepian, P. Springer Verlag, Berlin, 1968
"Mathematical Foundations of Network Analysis"
- 280 Stewart, W.J. C. ACM V.21 (1978), 144 - 152
"A comparison of numerical techniques in markov modeling"
- 281 Whitehead, E.G. Courant Inst. of Math. Science 1972
"Enumerative Combinatorics"
- 282 Whitehead, E.G. Courant Inst. of Math. Science 1973
"Combinatorial Algorithms"

Anhang AAllgemeine Bezeichnungen

$\mathbb{N}$	Menge der positiven ganzen Zahlen
$\mathbb{N}_0$	Menge der nicht-negativen ganzen Zahlen
$\mathbb{N}_n$	$\{i \mid i \in \mathbb{N} \wedge i \leq n\}$
$\mathbb{R}$	Menge der reellen Zahlen
$\mathbb{R}^+$	Menge der nicht-negativen reellen Zahlen

Seien V, U Mengen.

2^V Potenzmenge von V

$|V|$ Mächtigkeit von V

$\binom{V}{n}$ $\{X \mid X \subseteq V \wedge |X| = n\}$

$\emptyset$ leere Menge

$V \times U$ Kartesisches Produkt von V und U

$V^{\overbrace{n \text{-mal}}}$ $\underbrace{V \times V \times \dots \times V}_{m\text{-mal}}$

Δ_V $\{(a,b) \mid (a,b) \in V \times V \wedge a = b\}$ (d.h. die Gleichheitsrelation auf V)

Sei $B = \{b_1, \dots, b_m\} \subseteq 2^V$.

$\bigcup B$ $b_1 \cup b_2 \cup \dots \cup b_m$

$\text{GF}(2)$ endlicher Körper mit 2 Elementen

$\hat{=}$ definitionsgemäß gleich

$:=$ erhält den Wert (Symbol der Wertzuweisung)

$\gg$ sehr viel größer als

$\approx$ annäherungsweise gleich

$\square$ Ende des Beweises

/.../ Literaturangabe

$O(g_1, \dots, g_m)$ Sei $f: D \rightarrow \mathbb{R}$ eine Abbildung, wobei $D \subseteq \mathbb{N}^m$. Seien $g_1, \dots, g_m$ reellwertige Funktionen auf D . Die algorithmische Komplexität zur Berechnung von f wird als $O(g_1, \dots, g_m)$ bezeichnet, wenn es eine Konstante $c \in \mathbb{R}^+$ gibt, sodaß zur Berechnung von $f(X)$, $X = (x_1, \dots, x_m)$, höchstens $c(g_1(x_1) + \dots + g_m(x_m))$ Rechenschritte benötigt werden für jedes $X \in D$.

Anhang BGraphentheoretische Grundbegriffe

Ausgangspunkt der hier einzuführenden graphentheoretischen Grundbegriffe sind die in Definition 2.1.1 angegebenen endlichen schlichten Graphen: der endliche, ungerichtete Graph $G_U = (V, E)$ und der endliche, gerichtete Graph $G_D = (V, A)$.

Definition B.1

Sei $G_U = (V, E)$. Sei $x \in V$.

$d(x) \equiv |\{y \mid \{x, y\} \in E\}|$ heißt Grad von x .

Definition B.2

Sei $G_D = (V, A)$. Sei $x \in V$.

$d^+(x) \equiv |\{y \mid (x, y) \in A\}|$ heißt positiver Grad von x .

$d^-(x) \equiv |\{y \mid (y, x) \in A\}|$ heißt negativer Grad von x .

$d(x) = d^+(x) + d^-(x)$ ist der Grad von x .

Definition B.3

$x \in V$ heißt isolierter Knoten genau dann, wenn $d(x) = 0$.

Definition B.4

Sei $G_U = (V, E)$. Sei $l \in \mathbb{N}$. Sei $C_U = (v_1, \dots, v_l)$ eine nicht leere, endliche Folge.

- i) C_U heißt ungerichtete Kette in G_U , wenn $v_j \in V$, $j \in \mathbb{N}_l$ und $\{v_k, v_{k+1}\} \in E$ für $k \in \mathbb{N}_{l-1}$.
- ii) v_1 und v_l heißen Endknoten, $l - 1$ die Länge von C_U .
- iii) C_U heißt Pfad, wenn C_U eine ungerichtete Kette ist und aus $j \neq k$ ($j, k \in \mathbb{N}_l$) $v_j \neq v_k$ folgt.
- iv) C_U verbindet v_1 mit v_l , wenn C_U ein Pfad ist.
- v) C_U heißt Kreis, wenn $l > 3$, $v_1 = v_l$ und $(v_1, \dots, v_{l-1})$ ein Pfad ist.

- vi) G_U heißt vollständig genau dann, wenn es zwischen je zwei verschiedenen Knoten $x, y \in V$ genau eine Kante $\{x, y\}$ gibt (d.h. wenn $E = \binom{V}{2}$).
- vii) G_U heißt genau dann zusammenhängend, wenn es zwischen je zwei Knoten $x, y \in V$ eine ungerichtete Kette $(x, \dots, y)$ gibt.

Folgerung B.5

Sei $G_U = (V, E)$. G_U ist genau dann zusammenhängend, wenn zwischen je zwei Knoten ein Pfad existiert.

Definition B.6

Sei $G_d = (V, A)$. Sei $l \in \mathbb{N}$. Sei $C_d = (v_1, \dots, v_l)$ eine nicht leere, endliche Folge.

- i) C_d heißt gerichtete Kette (bzw. Kette) in G_d , wenn $v_j \in V$, $j \in \mathbb{N}_l$ und $(v_k, v_{k+1}) \in A$ (bzw. $(v_k, v_{k+1}) \in A \vee (v_{k+1}, v_k) \in A$) für $\forall k \in \mathbb{N}_{l-1}$.
- ii) v_1 heißt Anfangsknoten, v_l Endknoten, $l - 1$ die Länge von C_d .
- iii) v_l ist von v_1 aus erreichbar, wenn $(v_1, \dots, v_l)$ eine gerichtete Kette ist.
- iv) C_d heißt Bahn, wenn C_d eine gerichtete Kette ist und aus $j \neq k$ ($j, k \in \mathbb{N}_l$) $v_j \neq v_k$ folgt.
- v) C_d heißt Zyklus, wenn $l > 2$, $v_1 = v_l$ und $(v_1, \dots, v_{l-1})$ eine Bahn ist.
- vi) G_d heißt streng zusammenhängend (bzw. zusammenhängend) genau dann, wenn es zwischen je zwei Knoten $x, y \in V$ eine gerichtete Kette (bzw. Kette) $(x, \dots, y)$ gibt.

Definition B.7

Sei $G_U = (V, E)$. Sei $h \in V$.

Ein Teilgraph $G' = (V', E')$ von G_U heißt Zusammenhangskomponente von h in G_U (kurz, Z-Komponente von h in G_U) genau dann, wenn folgendes gilt:

- i) $V' \hat{=} \{x \mid (x=h) \vee ((x \neq h) \wedge ((h, \dots, x) \text{ ist ein Pfad}))\}$.
- ii) $E' \hat{=} \binom{V'}{2} \cap E$.

Definition B.8

Ein Baum ist ein zusammenhängender, ungerichteter Graph, der keinen Kreis enthält.

Definition B.9

Sei $G_d = (V, A)$.

i) G_d heißt Wurzelbaum genau dann, wenn folgendes gilt:

- (1) G_d ist zusammenhängend.
- (2) Es gibt ein $r \in V$, sodaß $d^-(r) = 0$.
- (3) $\forall x (x \in V \setminus \{r\} \implies d^-(x) = 1)$.

r heißt Wurzel von G_d . $x \in V$ heißt Blatt des Wurzelbaums, wenn $d^+(x) = 0$.

ii) G_d heißt Binärbaum genau dann, wenn

- (1) G_d ein Wurzelbaum ist, und
- (2) $\forall x (x \in V \implies d^+(x) = 0 \vee d^+(x) = 2)$.

Definition B.10

Sei $T = (V, A)$ ein Binärbaum, dessen Wurzel r ist.

Sei $(u, v) \in A$. Dann heißt u der Vater von v und v der Sohn von u .

Seien $u, v \in V$. Sei $(u, \dots, v)$ eine Bahn. Dann heißt u Vorgänger von v und v Nachfolger von u . Wenn $u \neq v$, dann heißt u echter Vorgänger von v und v echter Nachfolger von u .

Die Höhe h_u eines Knotens $u \in V$ ist die Länge der Bahn von r nach u . Die Höhe eines Binärbaums ist die maximale Höhe seiner Blätter.

Sei $s \in V$ kein Blatt. Dann hat s genau zwei Söhne. Man ordnet die Söhne des Knotens s von links nach rechts und bezeichnet den linken Sohn als Nein-Sohn und den rechten Sohn als Ja-Sohn von s .

Ein (binärer) Teilbaum $T' = (V', A')$ mit der Wurzel $s \in V'$ ist ein Binärbaum, bei dem gilt: $V' \subseteq V$, und alle Knoten $t \in V' \setminus \{s\}$ sind echte Nachfolger von s .

Ein (binärer) Teilbaum $T_l (T_r)$ heißt linker (rechter) Teilbaum von s , wenn die Wurzel von $T_l (T_r)$ der Nein-Sohn (der Ja-Sohn) von s ist.

$T = (V, A)$ heißt vollständig, wenn es ein $k \in \mathbb{N}$ gibt, sodaß für $\forall u \in V$ folgendes gilt:

- i) $h_u = k \iff u$ ist ein Blatt.
- ii) $h_u < k \iff u$ besitzt genau einen Ja-Sohn und genau einen Nein-Sohn.

Folgerung B.11

Sei $T = (V, A)$ ein Binärbaum mit der Wurzel r .

- i) Die Blätter von T haben keinen echten Nachfolger.
- ii) r besitzt keinen echten Vorgänger.
- iii) Die Höhe der Wurzel ist 0.
- iv) Ist $s \in V$ kein Blatt, dann besitzt s genau einen Ja-Sohn und genau einen Nein-Sohn.
- v) Sei $s \in V$. Es gibt genau eine Bahn von r nach s .
- vi) Ein vollständiger Binärbaum der Höhe k hat genau $2^{k+1} - 1$ Knoten.

Satz B.12

Für jeden endlichen, nicht leeren Baum $B = (V, E)$ gilt $|E| = |V| - 1$.

Beweis: Der Beweis dieses Satzes ist z.B. in /277/ angegeben. $\square$

Definition B.13

Sei U eine nicht leere Menge. Sei $R \subseteq U \times U$ eine Relation auf U .

R ist genau dann eine partielle Ordnung (Halbordnung) auf U , wenn gilt:

- i) $\forall x \in U: (x, x) \in R$. (reflexiv)
- ii) $(x, y) \in R \wedge (y, x) \in R \Rightarrow x = y$. (antisymmetrisch)
- iii) $(x, y) \in R \wedge (y, z) \in R \Rightarrow (x, z) \in R$. (transitiv)

R ist eine lineare Ordnung (totale Ordnung) auf U genau dann, wenn R eine partielle Ordnung auf U ist und der Bedingung

- iv) $\forall x \forall y (x \in U \wedge y \in U \Rightarrow (x, y) \in R \vee (y, x) \in R)$ (Vergleichbarkeit)

genügt. Sei R eine lineare Ordnung auf U .

Ein Element s heißt genau dann maximal in (U, R) , wenn gilt:

$$\forall x \in U: (s R x \Rightarrow x = s) .$$

Ein Element t heißt genau dann minimal in (U, R) , wenn gilt:

$$\forall x \in U: (x R t \Rightarrow x = t) .$$

Anhang C $P(n)$ und 2^n für $0 \leq n \leq 15$

<u>n</u>	<u>$P(n)^*$</u>	<u>2^n</u>
0	1	1
1	1	2
2	2	4
3	5	8
4	15	16
5	52	32
6	203	64
7	877	128
8	4.140	256
9	21.147	512
10	115.975	1.024
11	678.570	2.048
12	4.213.597	4.096
13	27.644.437	8.192
14	190.899.322	16.384
15	1.382.958.545	32.768

* $P(n)$ ist in /281/, S.11 angegeben.

D.2 Problem 2.8.1 - die K-Liste

v : 2 3 3 2 1 2 3 1 1 1 2 2 2 1 1 1 3 1 1 1 2 3 2 2 3 2
 1 2 0 4 5 4 6 2 2 3 5 0 1 8 2 6 8 9 5 5 6 7 8 9 0 1 7 3 3 3 4 4 7 1 9
 g_v : 8 8 8 8 7 6 6 6 5 5 5 5 5 5 4 4 4 4 4 3 3 3 3 3 3 3 3 3 2 2 2 2 2 1

i	s_i	w_i
1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1	50
2	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0	40
3	0 1 0 0 0 1 1 1	26
4	0 1 0 0 0 0 1 1	16
5	0 1 0 0 0 0 1 0	11
6	0 0 0 0 0 0 0 1 0 1 0 0	20
7	0 0 0 0 0 0 0 1 0 1 0	10
8	0 1 1 1	15
9	0 1 0 1	10
10	0 0 0 0 0 0 0 0 1 0 1 0	10
11	0 1 0 0 1 0 0 0 0 0 0 0 0 0	10
12	0 1 1 0 0 0 0 0 0 0 0 0 0 0	7
13	0 1 0 1 0 0 0 0 0 0 0 0 0 0	4
14	0 1 1 0 0 0 0 0 0 0 0 0 0	7
15	0 1 0 0 0 0 0 0 0 0 0 0 0 0	5
16	0 1 0 0 0 0 0 0 0 0 0 0 0 0	5
17	0 1 0 0 0 1 0 0 0	5
18	0 1 1	5
19	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0	4
20	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0	3
21	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0	2
22	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0	2
23	0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0	4
24	0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0	2
25	0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0	2
26	0 1 1 0 0 0 0 0 0 0 0 0 0 0 0	3
27	0 1 0 1 0 0 0 0 0 0 0 0 0 0 0	2
28	0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0	2
29	0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0	1
30	0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0	1
31	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0	2
32	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0	2
33	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0	2
34	0 1 0 0 0 0 0 0 0 0 0 0 0 0 0	2
35	0 1 0 0 0 0 0 0 0 0	1
36	0 1 1 0 0 0 0 0 0 0 0 0 0	1
37	0 1 0 1 0 0 0 0 0 0 0 0 0	1

0.3 Problem 2.8.1 - die schematische Darstellung des binären Verzweigungsprozesses

Das Schema	$\leq w_i$
$\emptyset \rightarrow 1 \rightarrow 5 \rightarrow 11 \rightarrow 15 \rightarrow 17 \rightarrow 19 \rightarrow 26 \rightarrow 29 \rightarrow 35$ $\downarrow$ $\rightarrow 30 \rightarrow 35$ $\downarrow$ $\rightarrow 28 \rightarrow 35 \checkmark$ $\downarrow$ $\rightarrow 29 \checkmark$ $\downarrow$ $\rightarrow 20 \rightarrow 26 \rightarrow 30 \checkmark$ $\downarrow$ $\rightarrow 21 \rightarrow 25 \rightarrow 26 \checkmark$ $\downarrow$ $\rightarrow 22 \checkmark$ $\downarrow$ $\rightarrow 23 \rightarrow 26 \rightarrow 30 \rightarrow 35$ $\downarrow$ $\rightarrow 19 \rightarrow 26 \rightarrow 29 \checkmark$ $\downarrow$ $\rightarrow 16 \rightarrow 17 \rightarrow 19 \rightarrow 26 \rightarrow 29 \rightarrow 35$ $\downarrow$ $\rightarrow 30 \rightarrow 35$ $\downarrow$ $\rightarrow 17 \rightarrow 19 \rightarrow 26 \rightarrow 29 \rightarrow 31 \checkmark$ $\downarrow$ $\rightarrow 19 \rightarrow 21 \rightarrow 29 \rightarrow 31 \rightarrow 32 \checkmark$ $\downarrow$ $\rightarrow 23 \rightarrow 26 \rightarrow 30 \rightarrow 31 \rightarrow 32 \checkmark$ $\downarrow$ $\rightarrow 12 \rightarrow 14 \rightarrow 15 \rightarrow 17 \rightarrow 19 \rightarrow 29 \rightarrow 35$ $\downarrow$ $\rightarrow 7 \rightarrow 12 \rightarrow 14 \rightarrow 15 \rightarrow 17 \rightarrow 19 \rightarrow 29 \rightarrow 34 \rightarrow 35$ $\downarrow$ $\rightarrow 16 \rightarrow 17 \rightarrow 19 \rightarrow 29 \rightarrow 34 \rightarrow 35$ $\downarrow$ $\rightarrow 10 \rightarrow 12 \rightarrow 14 \rightarrow 15 \rightarrow 17 \rightarrow 19 \rightarrow 29 \rightarrow 34 \rightarrow 35$ $\downarrow$ $\rightarrow 16 \rightarrow 17 \rightarrow 19 \rightarrow 29 \rightarrow 34 \rightarrow 35$ $\downarrow$ $\rightarrow 2 \rightarrow 4 \rightarrow 12 \rightarrow 14 \rightarrow 15 \rightarrow 17 \rightarrow 19 \rightarrow 29 \checkmark$ $\downarrow$ $\rightarrow 3 \checkmark$ $\downarrow$ $\rightarrow 4 \rightarrow 6 \rightarrow 12 \rightarrow 14 \rightarrow 15 \rightarrow 17 \rightarrow 19 \rightarrow 29 \checkmark$ $\downarrow$ $\checkmark$	 90 90 90 90 90 90 91 92 92 92 92

D.4 Problem 2.8.1 - die Lösungen

b	v ∈ b				G(b)	w(b)
	Lösung 1	Lösung 2	Lösung 3	Lösung 4		
b ₁	27, 28, 29	27, 28, 29	27, 28, 29	27, 28, 29	8	50
○ b ₂	26, 31	26, 31	32, 31	32, 31	8	10
b ₃	6, 7	6, 7	6, 7	6, 7	6	7
b ₄	8, 9	8, 9	8, 9	8, 9	6	7
b ₅	23, 24, 10	23, 24, 10	23, 24, 10	23, 24, 10	8	5
○ b ₆	17, 18	17, 19	17, 18	17, 19	7	5
b ₇	2, 3, 4	2, 3, 4	2, 4, 3	2, 3, 4	8	4
b ₈	33, 35	33, 35	33, 35	33, 35	7	2
b ₉	5, 12	5, 12	5, 12	5, 12	8	1
b ₁₀	11, 13	11, 13	11, 13	11, 13	8	1
b ₁₁	1	1	1	1	8	0
b ₁₂	22	22	22	22	8	0
b ₁₃	30	30	30	30	8	0
b ₁₄	34	34	34	34	8	0
○ b ₁₅	16, 19	16, 18	16, 19	16, 18	8	0
b ₁₆	25	25	25	25	7	0
b ₁₇	14	14	14	14	6	0
○ b ₁₈	32	32	26	26	6	0
b ₁₉	15	15	15	15	5	0
b ₂₀	20	20	20	20	5	0
b ₂₁	21	21	21	21	5	0

$\hat{Z} = \sum w(b) = 92$

$$\alpha = 21/19 .$$

D.5 Problem 2.8.2 - das Problem und die Lösungen

$$V = \{1, 2, \dots, 14\}.$$

$$g_i = 1 \text{ für } i \in \mathbb{N}_{14}.$$

$$a = 4.$$

$$E = \{[k, l] \mid f_{kl} > 0\}, \text{ dabei wird } f = (f_{kl})_{k, l \in V} \text{ folgendermaßen angegeben:}$$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1		2	0	0	0	0	0	0	0	0	0	0	0	0
2			42	0	0	40	0	0	0	0	0	0	0	0
3				42	0	0	0	0	0	0	0	0	0	0
4					41	0	0	0	1	0	0	0	0	0
5						41	0	0	0	0	0	0	0	0
6							1	0	0	0	0	0	0	0
7								101	100	0	0	0	0	0
8									101	0	0	0	0	0
9										2	0	0	0	0
10											22	20	0	0
11												22	0	0
12													2	0
13														2
14														

Die Lösungen:

$$\hat{Z} = 495.$$

$$B_1 = \{\{1, 2, 3, 4\}, \{5, 6\}, \{7, 8, 9\}, \{10, 11, 12, 13\}, \{14\}\}.$$

$$B_2 = \{\{1, 2, 3, 4\}, \{5, 6\}, \{7, 8, 9\}, \{10, 11, 12\}, \{13, 14\}\}.$$

$$\alpha = 5/4.$$

D.8 Problem 2.8.3

$a = 3$ (Seiten)

$V = \{1, 2, \dots, 29\}$.

Jeder Knoten $v \in V$ stellt eine Menge von Seiten dar:

$v \in V$	Seitenmenge
1	$\{14\}$
2	$\{14, 7\}$
3	$\{8, 7\}$
4	$\{14, 11\}$
5	$\{14, 10\}$
6	$\{11, 10\}$
7	$\{13, 10\}$
8	$\{10, 9\}$
9	$\{9, 8\}$
10	$\{15\}$
11	$\{15, 8\}$
12	$\{8\}$
13	$\{9\}$
14	$\{10, 8\}$
15	$\{10\}$
16	$\{10, 7\}$
17	$\{14, 13\}$
18	$\{14, 12\}$
19	$\{14, 9\}$
20	$\{14, 8\}$
21	$\{13, 11\}$
22	$\{12, 11\}$
23	$\{11, 7\}$
24	$\{11\}$
25	$\{14, 13, 11\}$
26	$\{13, 9\}$
27	$\{12, 9\}$
28	$\{12\}$
29	$\{13, 12\}$

D.8 Problem 2.8.3 (Fortsetzung)

Die Matrix des Programmgraphen (gerichteten Graphen)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
1	0	40	0	115	35	0	0	0	0	0	0	0	0	0	0	0	1258	0	1	0	0	0	0	0	0	0	0	0	0
2	0	0	1	102	35	0	0	0	0	0	0	0	0	0	0	4725	9	0	0	0	0	0	4882	1211	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
4	0	64	0	0	49	105	0	0	0	0	0	0	0	0	56	0	76211	0	1	0	45600	35770	728	0	0	5	9	0	25
5	0	0	0	0	0	99863	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
6	110	0	0	105	99709	0	27454	240	0	0	0	0	0	0	0	4760	0	0	0	1	90	0	1212	0	0	0	0	0	
7	0	0	0	0	0	32179	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
8	0	0	0	0	0	0	0	12636	0	0	0	0	0	18482	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
9	1	0	0	0	0	0	0	9272	0	261	0	0	0	1703	3274	0	0	0	2	5	0	0	0	0	0	0	0	0	
10	0	0	0	0	0	0	0	0	0	261	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
11	0	0	0	0	0	0	0	0	0	0	261	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
12	0	0	0	0	0	0	0	0	0	0	0	261	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
13	1	0	0	0	0	0	0	15200	153	0	18	0	0	0	90	0	0	0	0	0	0	0	0	0	0	0	0	0	
14	0	0	0	0	0	0	0	3282	1721	0	0	15200	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
15	9	0	0	0	0	1533	0	3124	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
16	0	4760	0	0	0	0	4725	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
17	0	43	0	76373	35	0	0	0	0	0	0	0	0	0	0	0	0	1024	0	0	0	21	0	0	0	0	0	0	
18	0	0	0	1024	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
19	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	9	9	0	0	66	57	0	0	
20	0	0	0	0	0	1	0	0	4	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
21	1326	0	0	79171	0	0	0	0	0	0	0	0	0	0	1211	0	0	0	22	0	0	961	2870	0	0	0	0	0	
22	0	0	0	961	0	0	0	0	0	0	0	0	0	0	0	0	18	0	0	0	35771	0	0	0	10	0	0	78	
23	0	6058	0	0	0	0	0	0	0	0	0	0	0	0	35	0	0	0	0	0	3634	0	0	0	0	0	0	0	
24	0	0	0	728	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	448	0	0	0	35	0	0	0	
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	35	0	0	0	0	0	
26	0	0	0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	114	0	0	0	0	0	0	0	0	2	0	
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	51	0	0	0	
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	
29	0	0	0	28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	77	0	0	0	0	0	0	

D.9 Problem 2.8.3 - die K-Liste (bezieht sich auf die Menge der Seiten)

	15	14	13	12	11	10	9	8	7 (Seitennummer)	
i	$\underline{s}_i$									w_i
1	0	1	1	0	1	0	0	0	0	281,265
2	0	1	0	0	1	1	0	0	0	202,417
3	0	1	0	1	1	0	0	0	0	36,731
4	0	1	0	0	0	1	1	0	0	18,460
5	0	1	0	0	1	0	0	0	1	11,834
6	0	1	0	0	0	1	0	0	1	9,604
7	0	1	1	0	0	0	1	0	0	1,440
8	0	1	1	0	0	0	0	0	1	1,350
9	0	1	1	0	0	1	0	0	0	1,337
10	0	1	1	1	0	0	0	0	0	1,024
11	0	1	0	0	1	0	0	0	0	843
12	0	1	0	0	0	0	1	1	0	448
13	0	1	0	0	1	0	1	0	0	118
14	0	1	0	1	0	0	1	0	0	52
15	0	1	0	0	0	1	0	0	0	44
16	0	1	0	0	0	0	1	0	1	42
17	0	1	0	0	0	0	0	1	1	42
18	0	1	0	0	0	0	0	0	1	40
19	0	1	0	0	0	0	1	0	0	2
20	0	0	0	0	0	1	1	1	0	84,416
21	0	0	0	0	0	1	1	0	0	18,414
22	0	0	0	0	0	1	0	1	0	18
23	0	0	1	0	1	1	0	0	0	62,915
24	0	0	1	1	1	0	0	0	0	37,335
25	0	0	1	0	1	0	0	0	1	6,952
26	0	0	1	0	0	1	0	0	1	4,725
27	0	0	1	0	1	0	0	0	0	448
28	0	0	1	1	0	0	1	0	0	55
29	0	0	0	0	1	1	1	0	0	20,187
30	0	0	0	0	1	1	0	0	1	7,540
31	0	0	0	0	1	1	0	0	0	1,533
32	1	0	0	0	0	0	1	1	0	1,215
33	1	0	0	0	0	0	0	1	0	522
34	1	0	0	0	0	1	0	1	0	279
35	0	0	0	0	0	0	1	1	0	279

D.10 Problem 2.8.3 - der binäre Verzweigungsprozeß und die Lösung

Die schematische Darstellung des binären Verzweigungsprozesses:

$$\emptyset \longrightarrow 1 \longrightarrow 20$$

$$\leq w_i = 365,681 \quad (= w_1 + w_{20})$$

$$\longrightarrow 21 \longrightarrow 33 \checkmark$$

$$\checkmark \longrightarrow 2 \longrightarrow 28 \checkmark$$

$$\checkmark \longrightarrow 3 \checkmark$$

Die Lösung:

$$\hat{Z} = 365,681 .$$

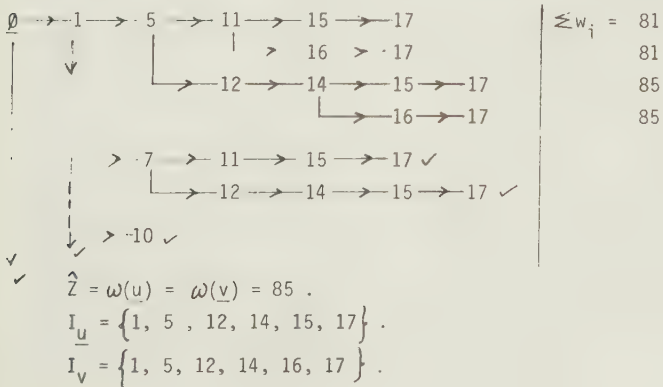
$$B = \left\{ \{14, 13, 11\}, \{10, 9, 8\}, \{15, 12, 7\} \right\} .$$

$$\alpha = 1 .$$

Bild E.1 : die 18-abgerundete Liste des Problems 2.8.1

[illegible][illegible]

Bild E.2 : die sich auf Bild E.1 beziehende Verzweigungsprozeß



Problem 2.8.2 - deterministische Approximation

Bild E.3: die 0,5-reduzierte Liste

$$v \in V : 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 0 \ 1 \ 2 \ 3 \ 4$$

i	s_i	w_i
1	0 0 0 0 0 0 0 1 1 1 1 0 0 0 0	304
2	0 0 0 0 0 0 0 1 1 1 0 0 0 0 0	302
3	0 0 0 0 0 0 1 1 1 1 1 0 0 0 0	303
4	0 0 0 1 0 0 1 1 1 1 0 0 0 0 0	303
5	0 0 1 1 0 0 0 1 1 1 0 0 0 0 0	144
6	0 0 1 1 0 0 1 0 1 0 0 0 0 0 0	143
7	0 0 1 1 1 1 1 0 0 0 0 0 0 0 0	124
8	0 0 1 1 1 0 0 0 1 0 0 0 0 0 0	84
9	0 0 1 1 1 0 0 0 0 0 0 0 0 0 0	83
10	0 0 0 0 1 1 1 1 1 0 0 0 0 0 0	143
11	0 0 0 0 1 1 1 0 1 0 0 0 0 0 0	142
12	0 1 0 0 0 1 1 1 1 0 0 0 0 0 0	142
13	0 1 0 0 0 1 1 0 1 0 0 0 0 0 0	141
14	0 1 1 1 1 0 0 0 0 0 0 0 0 0 0	125
15	0 1 1 1 0 1 0 0 0 0 0 0 0 0 0	124
16	0 1 1 0 1 1 0 0 0 0 0 0 0 0 0	123
17	0 1 0 1 1 1 0 0 0 0 0 0 0 0 0	122
18	0 1 1 1 0 0 0 0 1 0 0 0 0 0 0	85
19	0 1 1 1 0 0 0 0 0 0 0 0 0 0 0	84
20	0 1 1 0 0 1 1 0 0 0 0 0 0 0 0	83
21	0 1 1 0 0 1 0 0 0 0 0 0 0 0 0	82
22	0 1 0 0 1 1 1 0 0 0 0 0 0 0 0	82
23	0 1 0 0 1 1 0 0 0 0 0 0 0 0 0	81
24	0 0 0 0 0 0 0 0 1 1 1 1 0 0 0	125
25	0 0 0 0 0 0 0 0 1 1 1 0 1 0 0	123
26	0 0 0 0 0 0 0 0 1 1 1 0 0 0 0	103
27	0 0 0 0 0 0 0 0 1 1 0 0 0 0 0	101
28	1 1 1 1 0 0 0 0 0 0 0 0 0 0 0	86
29	1 1 1 0 0 1 0 0 0 0 0 0 0 0 0	84
30	1 1 0 0 1 1 0 0 0 0 0 0 0 0 0	83
31	1 1 1 0 0 0 0 0 0 0 0 0 0 0 0	44
32	1 1 0 0 0 1 1 0 0 0 0 0 0 0 0	43
33	0 0 0 0 0 0 0 0 0 1 1 1 1 0 0	66
34	0 0 0 0 0 0 0 0 0 0 1 1 1 1 0	66
35	0 0 0 0 0 0 0 0 0 0 1 1 1 0 0	64
36	0 0 0 0 0 0 0 0 0 0 0 1 1 1 1	26
37	0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0	24
38	0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0	22
39	0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1	4
40	0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0	2
41	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1	2

Bild E.4: der auf Bild E.3 bezogene VP

Das Schema	$\hat{Z}(w)$
$\emptyset \rightarrow 1 \rightarrow 7 \rightarrow 36$	454
$\downarrow \rightarrow 9 \checkmark$	
$\checkmark \rightarrow 14 \rightarrow 36$	455
$\rightarrow 2 \rightarrow 7 \rightarrow 34$	492
$\checkmark \rightarrow 9 \checkmark$	
$\rightarrow 14 \rightarrow 34$	493
$\rightarrow 3 \rightarrow 9 \checkmark$	
$\rightarrow 14 \rightarrow 34$	494
$\rightarrow 4 \rightarrow 16 \checkmark$	
$\checkmark$	
$\hat{Z}(w) = 494$	
$I_u = \{3, 14, 34\}$	

Problem von Christofides und Brooker (Beispiel 3.5.5)

Bild E.5 : die K-Liste

 $v \in V: 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8$

i	$\underline{s}_i$	w_i
1	1 1 0 0 1 1 0 0	370
2	1 1 1 0 1 0 0 0	290
3	1 1 1 0 0 1 0 0	270
4	1 1 1 1 0 0 0 0	250
5	1 1 0 1 1 0 0 0	240
6	1 1 0 0 0 1 0 1	240
7	1 0 1 1 0 0 0 1	240
8	1 1 0 0 1 0 1 0	220
9	1 0 1 1 0 0 1 0	220
10	1 0 1 1 1 0 0 0	210
11	1 0 1 0 1 1 1 0	170
12	1 0 0 1 0 1 0 1	160
13	1 0 1 1 0 1 0 0	150
14	1 0 0 1 1 1 1 0	140
15	1 0 0 0 1 1 0 1	140
16	1 0 1 0 0 1 0 1	120
17	1 0 0 0 1 1 1 0	120
18	1 0 1 0 1 0 1 0	110
19	1 0 0 0 0 1 1 1	110
20	0 1 0 1 1 1 0 0	320
21	0 1 0 0 1 1 0 1	320
22	0 1 1 0 1 1 0 0	310
23	0 1 0 0 1 1 1 0	300
24	0 1 1 1 1 0 0 0	270
25	0 1 0 1 1 0 1 0	260
26	0 1 0 1 1 0 0 1	250
27	0 1 1 1 0 0 0 1	200
28	0 1 1 1 0 0 1 0	180
29	0 1 0 0 1 0 1 1	180
30	0 1 1 1 0 1 0 0	170
31	0 1 1 0 1 0 1 0	170
32	0 1 1 0 0 1 0 1	160
33	0 0 1 1 0 0 1 1	310
34	0 0 1 1 1 0 1 0	240
35	0 0 1 1 1 0 0 1	230
36	0 0 1 1 0 1 0 1	230
37	0 0 1 1 1 1 0 0	210
38	0 0 1 0 1 1 0 1	150
39	0 0 1 0 1 1 1 0	130
40	0 0 1 0 1 0 1 1	100
41	0 0 0 1 1 0 1 1	310
42	0 0 0 1 1 1 0 1	280
43	0 0 0 1 0 1 1 1	280
44	0 0 0 1 1 1 1 0	240
45	0 0 0 0 1 1 1 1	210

Bild E.6: der auf Bild E.5 bezogene VP

Das Schema	$\sum w_i$
$\emptyset \rightarrow 1 \rightarrow 33$	680
$\downarrow \rightarrow 2 \rightarrow 43$	570

$$\hat{Z} = W(\underline{u}) = 680.$$

$$I_{\underline{u}} = \{1, 33\}.$$

Lebenslauf

Am 17 April 1936 wurde ich als Sohn des Lehrers Giok Boen Ang und dessen Ehefrau Peng Lian (geborene Lok) in Hokkian, China, geboren. Drei Monate danach wanderten meine Eltern nach Indonesien aus und ich wurde mitgenommen.

Ich besuchte von 1942 bis 1947 die Volksschule in Semarang, von 1948 bis 1950 die Mittelschule in Surabaya und von 1951 bis 1953 die Oberschule in Surabaya.

Nach der Reifeprüfung studierte ich von 1954 bis 1959 Nachrichtentechnik an der technisch-wissenschaftlichen Fakultät der Universität Indonesia (diese Fakultät in Bandung erhielt 1958 die Bezeichnung "Institut Teknologi Bandung"). Nach dem Diplom arbeitete ich von 1960 bis Sommer 1963 als Assistent der Abteilung Nachrichtentechnik des Institut Teknologi Bandung.

Vom 1.9.1963 bis 30.4.1968 beschäftigte ich mich bei der Firma Siemens AG, WWT / Werk für Datenverarbeitung, Abteilung Prüf- und Wartung, in München.

Seit 1.9.1968 bin ich bei Herrn Prof. Dr. W. Händler, Institut für mathematische Maschinen und Datenverarbeitung der Universität Erlangen-Nürnberg, wissenschaftlich tätig. Vom 1.8.1970 bis 30.9.1977 wurde ich als wissenschaftliche Hilfskraft bei demselben Institut beschäftigt und mit der Wahrnehmung der Dienstgeschäfte eines wissenschaftlichen Assistenten betraut.

In letzter Zeit sind folgende Arbeitsberichte des Instituts für
Mathematische Maschinen und Datenverarbeitung erschienen:

Band 10:

- Nr. 1 Hoffmann, W.:
Modelle und Analysen von Teilnehmerrechensystemen,
angewendet auf das Teilnehmersystem RESPOND unter
CD 3300 MASTER.
(Februar 1977)
- Nr. 2 Körber, K., Werzinger, G.:
Das Modell SIM-LAB.
Simulation eines klinisch-chemischen Laboratoriums.
(März 1977)
- Nr. 3 Altmann, W., Weber, D.:
Programmierungsmethodik.
(März 1977)
- Nr. 4 Maehle, E.:
Fehlertolerante Rechnerstrukturen.
(April 1977)
- Nr. 5 Beth, T.:
Kombinatorik: Theorie der Vertretersysteme.
(April 1977)
- Nr. 6 Mertens, P.:
Einige Hinweise zur technischen Gestaltung von
Bildschirmdialogen.
(Mai 1977)
- Nr. 7 Jäpel, D.:
Ein Begriffssystem zur Formulierung von Schutzaspekten
und seine Anwendung auf Programmsysteme.
(Juni 1977)
- Nr. 8 Brendel, W., Grötsch, E., Nagl, M., Schneider, H.-J.:
Graphersetzungs-systeme, Beiträge zur Theorie, Anwen-
dung und Implementierung.
(Juni 1977)
- Nr. 9 Rüdél, W.:
Darstellung und Vergleich abstrakter Modelle zur
Beschreibung geschützter Systeme.
(Juli 1977)
- Nr. 10 Bolch, B., Gall, R., Grassl, M., Schmidt, B.:
Modelle für Trommelspeicher.
(Juli 1977)
- Nr. 11 v. Eichhorn, K.:
Steiner Systeme, Codes und affine Räume.
(Juli 1977)

Band 10 (Forts.)

- Nr. 12 Göttler, H.:
Zweistufige Graphmanipulationssysteme für die
Semantik von Programmiersprachen.
(Juli 1977)
- Nr. 13 Keramidis, S.:
Ein allgemeines formales Modell für Betriebs-
systeme und dessen Implementierung.
(September 1977)
- Nr. 14 Arnold, F., Benker, L., Grütz, M.:
Die Fast Fourier Transformation und Möglich-
keiten ihrer Implementierung auf EGPA.
(November 1977)
- Nr. 15 Gardill, R., Händler, W., Hessling, H.,
Klar, R., Spies, P. P.:
ERES
Eine nichtprozedurale Rechnerentwurfssprache
mit präziser Zeitbeschreibung
(November 1977)
- Nr. 16 Wichmann, H.:
Entwicklung und Bewertung von Algorithmen für
ein mikroprogrammiertes Parallel- und Assoziativ-
konzept auf Mikroprogramm- und Benutzerebene.
(Dezember 1977)
- Nr. 17 Brügmann, H. W.:
EBIDIS - Strukturierte Bildprogrammierung im Dialog.
(Dezember 1977)
- Nr. 18 Jahresbericht 1977 der Informatik.
(März 1978)

Band 11
=====

- Nr. 1 Körber, K., Werzinger, G.:
 Das Modell SIM-LAB.
 Beschreibung des Grundmodells.
 (Februar 1978)
- Nr. 2 Körber, K., Werzinger, G.:
 Das Modell SIM-LAB.
 Beschreibung der Modellerweiterungen und der Strategien.
 (Februar 1978)
- Nr. 3 Körber, K., Werzinger, G.:
 Das Modell SIM-LAB.
 Listing der Quellprogramme.
 (Februar 1978)
- Nr. 4 Beth, Th.:
 On Resolutions of Steiner Systems.
 (März 1978)
- Nr. 5 Leeb, K.:
 Salami-Taktik beim Quader-Packen.
 Riess, W:
 Zwei Optimierungsprobleme auf Ordnungen.
 (April 1978)
- Nr. 6 Händler, W., Henning, W., Kleinöder, W., Kneissl, J.,
 Volkert, J.:
 EPIK - Erlanger Projekt Interaktive Kartographie.
 (April 1978)
- Nr. 7 Schreiber, H.:
 Hardware-Messung und Analyse des Ablaufgeschehens
 in Rechnerkernen.
 (April 1978)
- Nr. 8 Weber, D.:
 Datengraphen und deren Transformation:
 Ein Konzept zur Spezifikation von Datentypen.
 (Juni 1978)
- Nr. 9 Musielak, H., Schmidt, B., Stössel, M.:
 Vergleich von Simulationssprachen:
 GPSS-FORTRAN - GPSS - GASP IV.
 (Juli 1978)
- Nr. 10 Solymosi, A.:
 Synthese von analysierenden Automaten auf Grund
 von formalen Grammatiken.
 (Juli 1978)
- Nr. 11 Meyer, K., Schaller, K., Schmidt, B.:
 SIEM-QUEUE.
 Beschreibung eines Simulationsmodells für ein
 komplexes Warteschlangensystem, das mit Hilfe von
 Eingabedaten zusammengestellt werden kann.
 (Juli 1978)

Band 11 (Forts.)
=====

- Nr. 12 Bodendorf, F., Eckardt, T., Mertens, P., Schrammel, D.:
Benutzung und Beurteilung von Kleinrechnersystemen in
einer Hochschule - Ein Beitrag zur Dezentralisierungs-
diskussion.
(August 1978)
- Nr. 13 Rabinovitz, M.:
An Approach to a Reconfigurable Multiprocessor System.
Bode, A., Händler, W.:
Classification d'architectures parallèles:
Introduction de la notation ECS et application au
projet EGPA.
Sigmund, W.:
Parallel Machine Functions in Program Transformation
and Execution.
(September 1978)
- Nr. 14 Beth, Th., Strehl, V.:
Materialien zur Codierungstheorie.
(September 1978)
- Nr. 15 Wendler, K.:
Betriebssystemaspekte in hierarchisch modularen
Polyprozessorsystemen - Modellierungsansätze und
Koordinierungsmechanismen.
(Oktober 1978)
- Nr. 17 Hoffmann, W.:
Warteschlangenmodelle für die Parallelver-
arbeitung.
(November 1978)
- Nr. 18 Stöth, G.:
Höhere Programmiersprachen für Kleinrechner:
Anforderungen, Entwicklungskriterien, Compiler-
aspekte.
(November 1978)
- Nr. 19 Boder, A., Händler, W.:
Rechnerarchitektur - Grundlagen und Verfahren.
(November 1978)
- Nr. 20 Kneissl, F.:
Grundsoftware für die Textverarbeitung.
(Dezember 1978)
- Nr. 21 Leidel, W.:
Synchronisationsprobleme in Automatennetzen.
(Dezember 1978)

Band 11 (Forts.)
=====

- Nr. 22 Bodendorf, F., Renninger, W., Schaller, K.:
 Zwei neue Methodenbanksysteme für statistische
 Anwendungen.
 (Dezember 1978)
- Nr. 23 Jahresbericht 1978 der Informatik.
 (März 1979)

- Nr. 1 Elzer, P.:
Strukturierte Beschreibung von Prozesssystemen.
(Februar 1979)

- Nr. 2 Wunderatsch, H.:
Deterministisches Zuordnen in heterogenen
Prozessorsystemen mit Pseudo-Boole Methoden.
(Juni 1979)

- Nr. 3 Bunke, H.:
Sequentielle und parallele programmierte
Graph-Grammatiken.
(Juni 1979)

- Nr. 4 Luft, A. L.:
Zur ingenieurwissenschaftlichen Theorie von
Rechensystemen.
(August 1979)

- Nr. 5 Schmidt, B.:
Das Simulationsmodell SIM 330 - Einführung.
(August 1979)

- Nr. 6 Gardill, R.:
SKOP, Syntaxnotation als Kontrollstruktur
für Programmiersprachen.
(September 1979)

- Nr. 7 Keramidis, S.:
Entwurf und Implementierung eines Rechnerver-
bundnetzes für die Fertigungssteuerung.
Fallstudie zur Kontrolle zuverlässiger Systeme.
(September 1979)

